Int. J. Appl. Math. Comput. Sci., 2007, Vol. 17, No. 4, 515–537
DOI: 10.2478/v10006-007-0043-y

amcs

OBJECT LIBRARY OF ALGORITHMS FOR DYNAMIC OPTIMIZATION PROBLEMS: BENCHMARKING SQP AND NONLINEAR INTERIOR POINT METHODS

JACEK BŁASZCZYK *, ANDRZEJ KARBOWSKI *,**, KRZYSZTOF MALINOWSKI *,**

* Institute of Control and Computation Engineering
Faculty of Electronics and Information Technology
Warsaw University of Technology
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: J.Blaszczyk,A.Karbowski,K.Malinowsk@ia.pw.edu.pl

** Research and Academic Computer Network (NASK)
ul. Wąwozowa 18, 02-796 Warsaw, Poland

The main purpose of this paper is to describe the design, implementation and possibilities of our object-oriented library of algorithms for dynamic optimization problems. We briefly present library classes for the formulation and manipulation of dynamic optimization problems, and give a general survey of solver classes for unconstrained and constrained optimization. We also demonstrate methods of derivative evaluation that we used, in particular automatic differentiation. Further, we briefly formulate and characterize the class of problems solved by our optimization classes. The solution of dynamic optimization problems with general constraints is performed by transformation into structured large-scale nonlinear programming problems and applying methods for nonlinear optimization. Two main algorithms of solvers for constrained dynamic optimization are presented in detail: the sequential quadratic programming (SQP) exploring the multistage structure of the dynamic optimization problem during the solution of a sequence of quadratic subproblems, and the nonlinear interior-point method implemented in a general-purpose large-scale optimizer IPOPT. At the end, we include a typical numerical example of the application of the constrained solvers to a large-scale discrete-time optimal control problem and we use the performance profiles methodology to compare the efficiency and robustness of different solvers or different options of the same solver. In conclusions, we summarize our experience gathered during the library development.

Keywords: dynamic optimization, large-scale optimization, sequential quadratic programming, nonlinear interior-point methods, object-oriented numerical computations, automatic differentiation, performance data analysis.

1. Introduction

The goal of dynamic optimization is to determine optimal control and state trajectories for a dynamic system over a finite horizon such that a certain performance criterion is minimized. In generally available literature we can find several, more or less complicated, algorithms of dynamic optimization, but many of them have not been implemented in optimization packages yet, and are available only for a narrow group of optimal control experts. One of the main objectives of developing our dynamic optimization library was the wish to popularize both older, already verified algorithms, and recent, modern methods for the computation of optimal control. We also hope that our library will establish reusable and extendible environment for easy implementation and the testing of new optimal control algorithms. Another purpose of our work was practical verification of the object methodology in the design and implementation of large and complex numerical libraries, and also the application of automatic differentiation to compute derivative values in optimization problems.

A detailed description of unconstrained and constrained parts of our library was given in the conference presentations (Błaszczyk *et al.*, 2002a; Błaszczyk *et al.*, 2002b; Błaszczyk *et al.*, 2003). Here we briefly present the possibilities of the library, two main optimization solvers for constrained problems, their application

to a typical example of large-scale discrete-time optimal control problem and, finally, a benchmark of solvers by the application of performance profile methodology.

2. Description of the Dynamic Optimization Library

Our object-oriented numerical library was designed for the solution of discrete-time optimal control problems. Dynamic optimization problems are specified in the programming language C++ with the use of the appropriate classes for their formulation, and automatic differentiation (provided by the ADOL-C library) is used for the automatic calculation of exact first and second derivatives. Special emphasis is put on the treatment of dynamic optimization problems with large-scale nonlinear programming algorithms, such as sequential quadratic programming and a nonlinear interior-point method. The implementation of all classes for problem formulation and solving is based upon dense matrix codes of the NEWMAT and UBLAS libraries for matrix computations in C++, which were extended for additional factorization procedures, classes for the management of sparse-block matrices, and converter functions for different sparse matrix formats needed by sparse matrix solvers. The possibility of replaceable usage of two matrix libraries is provided by the template mechanism of the C++ language. The configuration of parameters of optimization solvers and controlling their execution are possible with the configuration files (in the ini-file format) and set/get methods of computational classes.

As the implementation language for our library we chose the C++ programming language. From a numerical programmer's point of view, C++ offers a wide set of facilities, e.g., the possibility to put numerical formulas in a program in natural syntax using objects from the mathematical world. The C++ language has also features that are ideal to design and implement large software libraries. An important criterion for the choice of the C++ language was the availability of a large set of numerical libraries, often under a public domain status or free for educational purposes. Among other things, the implementation of our library has been based on the following C++ libraries:

GNU libstdc++ standard C++ library, available at `http://www.gcc.gnu.org/libstdc++/`

ADOL-C automatic differentiation library by A. Griewank from the Technical University of Dresden, available at `http://www.math.tu-dresden.de/~adol-c/`

NEWMAT matrix library by R. Davies from New Zealand, available at `http://www.robertnz.net/`

UBLAS matrix library from the BOOST project, available at `http://www.boost.org/`

The exception mechanism of the C++ language that was used in the library implementation allows the user to localize typical errors in the programs quickly and effectively (e.g., disallowed data conversion, a wrong value of a method parameter, exceeding the index range for a table, vector or matrix type), and also to handle errors in the optimization routines (e.g., an iteration limit, the lack of algorithm convergence). Then, the application of the C++ namespace mechanism excluded potential clashes of variable, function, class and object names, used in our library, with other libraries.

With the library, several computational examples of dynamic optimization problems are provided that allow a potential user to quickly become familiar with methods of new problem definition and with calling for optimization routines.

Besides the possibility of effective and exact solving of dynamic optimization problems, we wanted to achieve a modular construction of the library. Such a structure, supported by the class mechanism of the C++ language, allows the library user to experiment with different algorithms and to construct new algorithms as well. It is possible to distinguish three categories of classes/functions in such a designed library:

1. Classes for the formulation of dynamic optimization problems, the manipulation of their parameters and for simulation,
2. Computational classes, including primarily optimization routines,
3. Auxiliary classes and functions, for example, extending the capabilities of the NEWMAT and UBLAS matrix libraries, introducing new "base" data types, different converting functions.

2.1. Classes for Problem Formulation. For handling dynamic optimization problems formulated by library users we designed a special object interface. It allows the user to input all required data to formulate the problem: the dimensions of state and control vectors, the number of stages in the control horizon, the state equation function, the cost function per stage and the final cut function. For constrained problems we have special classes for handling stationary or nonstationary simple bounds on control values. An object of the class representing dynamic optimization problems may be manipulated in many ways, e.g., there is a possibility to change the initial values of problem parameters, to get their values and, finally, to carry out a simulation. In our library we anticipated the possibility to change the values of the parameters of optimization methods without the recompilation of programs – by acquiring the parameters from adequate configuration files. The library also provides the procedures to compare the values of functions and their

derivatives for two dynamic optimization problems, on the whole control horizon.

The library of classes for the formulation of dynamic optimization problems includes: (1) classes for unconstrained problems with derivatives computed by finite differences and with automatic differentiation, (2) classes for simple bounds constraints for stationary and nonstationary problems, (3) classes responsible for general inequality and equality constraints in dynamic optimization problems, with derivatives obtained from finite-difference approximations and with automatic differentiation, and, finally, (4) classes for constrained nonlinear problems inherited from the previous classes.

2.2. Solver Classes. One of fundamental design objectives was to provide the library with consistent and logical interface for optimization methods. The root of all computational classes is the abstract base class that defines the common subset of operations for optimization classes such as the management of the iteration process (the parameters of stopping criteria, the verbosity of the solver), the initialization of default parameters for solvers, saving intermediate and final results of optimization. From that class we inherited specific computational classes or indirect auxiliary classes. Some optimization methods, i.e., solvers based on the SQP algorithm, form a subhierarchy of classes.

From among a wide variety of numerical methods for dynamic optimization described in the literature, we chose those that fulfil the criteria of usefulness, robustness and good documentation of an algorithm. For the solution of unconstrained problems, we implemented direct methods for optimal control – the group of optimization algorithms classes based on the computation of the reduced gradient of a functional with respect to the control according to their description in (Findeisen *et al.*, 1980; Wierzbicki, 1984), including the simple gradient method in the control space (with Fortuna's modification), the conjugate directions method in the control space (with Fortuna's modification), with three types of conjugate direction computations (Fletcher-Reeves, Polak-Ribière, Sinnot-Luenberger), the variable metric method in the control space (with Fortuna's modification), and the Newton method in the control space (with Picard's modification). The second group of unconstrained method classes provided by our library is based on algorithms using a dynamic programming technique: the differential dynamic programming (DDP) method (with the shifting of the Hessian eigenvalues to promote global convergence), and a stagewise Newton method (also with shifting), both described in (Yakowitz and Rutherford, 1984; Pantoja, 1988).

On the other hand, for problems with simple constraints on control we implemented a class for projected descent minimization (PDMIN)—a robust and highly effective (especially for large-scale problems) version of the projection algorithm from (Schwartz and Polak, 1997), which extends the Bertsekas projection method described in (Bertsekas, 1982). The PDMIN class contains three versions of the descent direction method: steepest descent, Polak–Ribière conjugate gradient and limited memory BFGS.

The constrained part of the our library contains a wide range of optimization classes for the end user (mostly based on some version of the SQP algorithm). Firstly, we implemented the traditional penalty function algorithm for constrained dynamic optimization problems (Findeisen *et al.*, 1980; Wierzbicki, 1984). This algorithm solves the resulting auxiliary unconstrained dynamic optimization problems with the use of methods implemented in the unconstrained part of our library (simple or conjugate gradient methods with Fortuna's modification, variable metric, second variation, DDP, the stagewise Newton method). Further, there is a solver for problems with general constraints based on the SQP algorithm of the Powell type (Powell, 1978) and with the treatment of the resulting convex quadratic subproblems with an interior-point method. The main part of the SQP algorithm is the QP solver whose efficient implementation should explore the dynamic optimization problem structure and sparsity. Currently, three different QP classes are available with our library: (1) `Mehrotra`, which implements Mehrotra's primal-dual predictor-corrector method well-known in the literature for linear programming (Mehrotra, 1992), (2) `Franke`, Mehrotra's algorithm modified by R. Franke from the Technical University of Ilmenau (Franke, 1994) with the aim to improve its performance and robustness, and (3) `Gondzio`, the multiple centrality corrector variant of the primal-dual interior-point method for convex QP (Gondzio, 1994). The main computational effort of the interior-point method is the solution of sparse, symmetric and indefinite large systems of linear equations generating the Newton steps. Here the sparsity structure of the dynamic optimization problem is exploited. For that purpose we implemented a matrix solver developed by E. Arnold from the Technical University of Ilmenau, which performs a block-wise elimination. The algorithm is an extension of the Ricatti recursion for unconstrained linear-quadratic optimal control problems (Arnold and Puta, 1994; Arnold *et al.*, 1994). We based the implementation of our LQ-DOCP class on C++ codes from the HQP solver (Franke, 1998) and freely available from `http://hqp.sourceforge.net`. To compare the performance of the LQ-DOCP matrix solver with other general-purpose sparse direct solvers, we implemented interface classes to the sparse direct solvers MA27, MA47, WSMP, MUMPS, PARDISO, BLKFCLT, UMFPACK, and the iterative solvers GMRES, SYMMLQ, MINRES, SQMR. They are used to solve sparse symmetric indefinite systems of equations during iterations of the specialized QP

solvers. The specialized LQ-DOCP method outperforms clearly the other matrix solvers with respect to computational time. In our library we have a special version of Powell's SQP method optimized for constrained problems with only simple bounds on state and control. Apart from Powell's SQP method there is a solver for problems with general constrains based on the SQP algorithm of Schittkowski's type (Schittkowski, 1983) and with an interior-point QP solver. Like for Powell's SQP method there is a version of Schittkowski's SQP method tailored for problems with only simple bounds constraints. On the other hand, we also implemented Powell's SQP method for constrained problems with an active-set strategy for the solution of QP problems. The main step of this method is the solution of a linear-quadratic dynamic optimization problem with linear inequality constraints using the active-set strategy of Gill-Murray and Bertsekas, or Goldfarb-Idnani, and a Riccati-type solution algorithm for equality constrained QP subproblems. Finally, in our library we provide solvers which transform dynamic optimization problems into structured large-scale nonlinear programming problems and apply to their solution general NLP solvers. We implemented interfaces to NLP solvers based on various versions of the SQP algorithm (CFSQP, DONLP2, HQP, LOQO, KNITRO) or a nonlinear interior-point algorithm (IPOPT, KNITRO). The efficiency of such general-purpose NLP solvers used for the solution of dynamic optimization problems may be comparable (but not better) to specialized SQP solvers.

All computational codes may use automatic differentiation for the automatic calculation of exact first and second order derivatives or approximate derivatives needed by finite differences. The implementation of our computational classes allows the flexible exchange of a single algorithm's modules and their configuration.

2.3. Computation of Derivative Values. The requirement of derivative calculations in computational algorithms for dynamic optimization necessitates the application of numerical differentiation methods. For our library, in the first attempt, we implemented the computation of gradients, Jacobians and Hessians for dynamic optimization problem with the use of finite differences. The user of the library has an ability to choose between three versions of numerical differentiation:

- forward finite differences,
- backward finite differences,
- central finite differences.

The above methods of numerical differentiation have one big disadvantage—they provide only approximate derivatives. Fortunately, there is a method of automatic differentiation, which allows us to determine the derivatives of arbitrary order exactly and effectively. In the implementation of our library, we use for this purpose the well-known ADOL-C library of automatic differentiation of C/C++ programs (Griewank *et al.*, 1999). In the classes for problem formulation with the calculation of derivatives by automatic differentiation we use latest capabilities of the ADOL-C library, such as the computation of sparsity patterns and availability to compute the Jacobians and Hessians as sparse matrices. In order to calculate the derivatives by automatic differentiation in a dynamic optimization problem, the users of our library have to make only small changes in the definition of the problem functions and to use the proper classes for their problem formulation.

3. Discrete–Time Optimal Control Problem

The library was designed to solve the nonlinear (constrained or unconstrained) discrete-time optimal control problem which is stated as follows:

$$\text{(DOCP)} \quad \min_u J = \sum_{k=0}^{N-1} f_0^k(x^k, u^k) + F^N(x^N), \quad (1)$$

$$f_0^k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}, \quad F^N : \mathbb{R}^n \mapsto \mathbb{R},$$

with the state equation

$$x^{k+1} = f^k(x^k, u^k), \quad k = 0, \ldots, N-1, \quad x^0 = \bar{x}^0, \quad (2)$$

$$f^k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n,$$

and with trajectory constraints:

$$g^k(x^k, u^k) \geq 0, \quad k = 0, \ldots, N-1, \quad (3)$$
$$g^N(x^N) \geq 0,$$

$$g^k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^{r^k}, \quad g^N : \mathbb{R}^n \mapsto \mathbb{R}^{r^N},$$

where u^k and x^k are vectors of control and state variables, respectively, k is the stage number and N is the length of the control horizon. We assume that the functions F^N, f_0^k, f^k, g^k and g^N have continuous second derivatives. Also it is assumed that constraints may contain fixed initial or final states, as well as simple bounds for state and control variables and, in general, linear and nonlinear constraints of a mixed or a homogeneous type. The solution to the DOCP is the optimal control trajectory:

$$u^{[0,N-1]} = \{u^0, u^1, \ldots, u^{N-1}\}$$

and the optimal state trajectory, which results from the solution of the state equations.

For such a problem of dynamic optimization, we assume the following design specifications:

1. Problems with a finite control horizon are considered.

2. The constraints $g^k(\cdot,\cdot)$, $g^N(\cdot)$ may be inequalities or equalities, and only on state variables or mixed.
3. The objective function may have general Bolza form as given in Eqn. (1), and its two special cases are also considered:
   - without the final state function (the Lagrange form):
   $$F^N(\cdot) \equiv 0,$$
   - without the state cost function (the Mayer form):
   $$f_0^k(\cdot,\cdot) \equiv 0, \quad k = 0,\ldots,N-1.$$
4. The problem may be stationary or nonstationary.
5. Minimal-time problems are supported by their transformation to finite horizon problems.
6. The initial state x^0 is fixed (for some methods we will have a possibility to solve problems with free initial state).
7. The problem may contain a fixed final state.
8. The constrained problems with only simple, box bounds on state and control:
$$x_L^k \le x^k \le x_U^k, \quad k = 0,\ldots,N, \tag{4}$$
$$u_L^k \le u^k \le u_U^k, \quad k = 0,\ldots,N-1 \tag{5}$$
should be supported by specialized solvers.
9. Optimization problems may be large (especially for discretized continuous-time problems), in particular with $N = 1000$, $N = 10000$, and even $N = 100000$.

The characteristic features of the DOCP from the point of view of optimization methods are as follows:

- A large number of variables and constraints.
- Sparse and structurized derivative matrices for the objective function and constraints.
- A large computational cost of the objective function and constraints, and their derivatives.
- At the solution many constraints are active.

4. Main Algorithm for Constrained Problems

The solution of dynamic optimization problems with general constraints is performed by transformation into a structured large-scale nonlinear programming problem and applying efficient numerical optimization methods such as various versions of sequential quadratic programming or the nonlinear interior-point method.

4.1. Large-Scale Nonlinear Problem. We can consider the DOCP as a structured, large-scale nonlinear programming problem in decision variables (x^k, u^k). The state equations are transformed into equality constraints, and the trajectory constraints into inequality restrictions. Such transformations allow the application of general-purpose, efficient and robust NLP algorithms to the solution of the DOCP.

Let us define

$$y = \begin{bmatrix} x^0 \\ u^0 \\ \vdots \\ x^N \end{bmatrix}, \quad g(y) = \begin{bmatrix} g^0(x^0,u^0) \\ g^1(x^1,u^1) \\ \vdots \\ g^N(x^N) \end{bmatrix},$$

$$h(y) = \begin{bmatrix} h^0(y) \\ h^1(y) \\ \vdots \\ h^{N-1}(y) \end{bmatrix} = \begin{bmatrix} f^0(x^0,u^0) - x^1 \\ f^1(x^1,u^1) - x^2 \\ \vdots \\ f^{N-1}(x^{N-1},u^{N-1}) - x^N \end{bmatrix},$$

where $y \in \mathbb{R}^{m_y}$, $h \in \mathbb{R}^{m_h}$, $g \in \mathbb{R}^{m_g}$, $m_y = N(m+n)+n$, $m_h = Nn$, $m_g = \sum_{k=0}^{N} r^k$.

As can be seen, the discrete-time state and control variables for all stages are assembled in one large vector y of decision variables. We can rewrite the DOCP as the following large-scale nonlinear programming problem:

$$\text{(NLP)} \quad \min_y \{J(y) \mid h(y) = 0,\ g(y) \ge 0\}. \tag{6}$$

We assume $J(y)$, $h(y)$ and $g(y)$ to be twice continuously differentiable.

4.2. SQP Method. For the numerical solution of the DOCP transformed to the NLP problem (6), we use first of all sequential quadratic programming (SQP) algorithms. SQP methods are now standard in constrained nonlinear programming (Powell, 1978; Fletcher, 1987; Schittkowski, 1980), often for large-scale problems, due to their efficiency (a small number of the required objective function and gradient evaluations) and robustness.

In the SQP method we reduce the optimization of NLP problems to the solution of a sequence of local linear-quadratic programming (QP) problems. Such a substitution of a difficult problem for a sequence of simpler QP problems, resulting from local approximation of the objective function and constraints, is characteristic for constrained NLP methods. In the basic formulation, the SQP algorithm solves the NLP problem by a sequence of linear-quadratic approximations (QP problems) obtained by substituting linear approximations for nonlinear constraints (by expansion in a first-order Taylor series) and by substituting of the nonlinear objective function for its expansion in a second-order Taylor series, extended with

second-order information about constraints. The SQP algorithm has a superlinear rate of convergence, and the SQP version with exact calculation of the Lagrangian Hessian (with the use of second derivatives) is locally quadratically convergent. The convergence of the SQP algorithm from any starting point is obtained due to the application of a merit function during the calculation of a new approximation of the NLP solution. Typically, the merit function is a penalty function of a linear combination of the objective function and a constraint violation. The approximation of Lagrange multipliers of the NLP problem resulting from solution of the QP subproblem in the current iteration is used for the calculation of the update of the second-order part of the QP problem in the next iteration—it may be an analytical Hessian of the Lagrange function or its approximation determined by the modified BFGS or SR1 methods. The SQP methods has a two-level structure of iterations. During the major iteration we determine the next approximation of the NLP solution (with approximations of Lagrange multipliers for constraints) resulting from the solution of the QP subproblem. In minor iterations we simply iterate the solution of the QP problem.

The SQP algorithm in the basic form given by Powell in (Powell, 1978) is relatively simple to implement. "It (the SQP solver) can be programmed in an afternoon if one has a quadratic programming subroutine available ...", writes Powell. A classical SQP algorithm is usually quite effective for small problems and for problems in which most of computational time is occupied by the calculation of functions and gradient values. However, for larger problems computational time may be dominated by matrix calculations. To solve effectively large problems the part of SQP algorithm responsible for the solution of QP subproblems should take into consideration the structure of the DOCP. Also, the update procedure of the Hessian, present during every SQP iteration, should explore its block-diagonal sparsity structure. For large-scale problems, QP subproblems are solved by means of sparse matrix algebra techniques.

The key issue in the SQP algorithm is effective solution of the QP subproblem. Generally, there are two groups of QP methods: classical active-set (AS) strategies and modern interior-point (IP) methods. AS methods are quite good for small and medium problems, because of their non-polynomial computational complexity. The most popular active-set strategy is the simplex algorithm for linear programming. On the other hand, we have IP methods with a polynomial bound of computational complexity. IP algorithms are based on the idea of identifying active inequality constraints numerically, with the help of a nonlinear barrier function, and solving the resulting large-scale nonlinear equation system with a Newton method (the solution of a sequence of large-scale sparse symmetric indefinite systems of linear equations). IP methods show excellent numerical properties for large-scale QP problems. Typically, implementations of AS methods are based on dense matrix algebra, and IP methods require sparse matrix computations.

The special structure of the DOCP can be used during the solution of the QP subproblem with linear equality constraints based on an extension of the well-known Riccati solution procedure for unconstrained linear-quadratic control problems to the equality constrained case. Hence we obtain an effective and robust algorithm for this special optimal control problem.

4.2.1. SQP Solver. The implementation of the SQP solver with an interior-point algorithm for the solution of QP subproblems, for our dynamic optimization library, is based on the code of the HQP[1] solver implemented by R. Franke, E. Arnold and H. Linke, see (Franke and Arnold, 1999). The original contributions of this article's authors are: a new design of the SQP solver class hierarchy and the implementation of additional configurable modules for the SQP solver (different QP solvers, matrix solvers and Lagrangian Hessian approximations). Also, we changed the implementation of matrix algebra operations using the NEWMAT and UBLAS dense matrix libraries, and also the native matrix classes `SparseBlockMatrix` (for the management of QP problem matrices) and `SymmetricBlockDiagonalMatrix` (for the management of Lagrangian Hessian approximation). By those modifications, we obtained noticeable performance and accuracy improvements for large-scale problems in relation to the HQP solver code.

The solution of the NLP problem (6) is approached through the Lagrange function

$$L(y, p, \lambda) = J(y) - p^T h(y) - \lambda^T g(y), \qquad (7)$$

where $p \in \mathbb{R}^{m_h}$ and $\lambda \in \mathbb{R}^{m_g}$ are the Lagrange multiplier vectors for the equality and inequality constraints, respectively. The Lagrange-Newton-type SQP algorithm attempts to find a stationary point $(y_\star, p_\star, \lambda_\star)$ of the Lagrange function, with the help of solutions of local, linear-quadratic, approximations to the NLP problem (6).

In each SQP iteration i the search direction s^i for the improving in the given iterate y^i of y is obtained by solving the following local linear-quadratic approximation to the NLP problem:

$$\min_s \frac{1}{2} s^T H_i s + \nabla_y J(y^i)^T s \qquad (8)$$

subject to

$$\nabla_y h(y^i) s + h(y^i) = 0, \qquad (9)$$

$$\nabla_y g(y^i) s + g(y^i) \geq 0. \qquad (10)$$

[1] An open source C++ implementation of HQP is available at `http://hqp.sourceforge.net/`

The above convex QP problem is obtained by quadratic approximation of the Lagrange function (7) and local linearization of (6) at a given iterate y^i. The solution s^i of (8)–(10) is used to obtain an improved iterate y^{i+1}:

$$y^{i+1} := y^i + \alpha s^i, \tag{11}$$

where the step length α is obtained by minimizing an exact penalty function with the Armijo-type line-search. The required derivatives (the gradient ∇J, the Jacobians ∇h and ∇g) are obtained by automatic differentiation procedures available in the ADOL-C library.

For our dynamic optimization library three SQP solver classes were implemented based on Powell algorithms (Powell, 1978), with the solution of QP subproblems by an active-set strategy or an interior-point method, and that of Schittkowski (Schittkowski, 1983), with an interior-point method. In general, we would recommend Powell's algorithm, which in our applications appeared very robust when solving large-scale problems. The implementation of Powell's SQP algorithm has been extended with a watchdog strategy (Chamberlain *et al.*, 1982) to improve further the robustness. Schittkowski's algorithm sometimes provides advantages for highly nonlinear problems.

Because of possibly high dimensions of the optimization problems, it is essential to take into account the sparsity structure of the problem in the solution procedure:

- the gradient ∇J and the Jacobians ∇h and ∇g are calculated taking into account the stage-wise DOCP problem formulation in (2) and (3),
- the block-diagonal structure of the Hessian $\nabla^2_{yy} L$ is preserved during the update procedure of its approximation, and
- because of its advantageous numerical behavior, especially for large-scale problems, an interior-point method is preferred to be used for the solution of the quadratic programming problem (8)–(10).

The quadratic approximation of the Lagrangian (7) is probably most for crucial to advantageous application of SQP solvers to large-scale problems. First of all, the matrix H_i must be positive definite, to get a convex quadratic subproblem, which can be treated efficiently by the QP solver. Secondly, H_i should be sufficiently sparse in order to allow efficient application of sparse matrix solvers. Sparsity requirement in the case of (DOCPs) is fulfilled by the analytical Lagrangian Hessian, which exhibits the following block-diagonal structure:

$$H_{i+1} = \nabla^2_{yy} L(y^{i+1}, p^i, \lambda^i) = \begin{bmatrix} \square & \square & & & & \\ \square & \square & & & & \\ & & \ddots & & & \\ & & & \square & \square & \\ & & & \square & \square & \\ & & & & & \square \end{bmatrix}. \tag{12}$$

This structure should be preserved during the Hessian update procedure. Positive definiteness of the above matrix may be guaranteed by the Gerschgorin modification:

$$h_{i,i} := \max \left\{ \epsilon + \sum_{j \neq i} |h_{i,j}|, h_{i,i} \right\}, \quad i, j = 1, \ldots, m_y, \tag{13}$$

but this often results in rather poor convergence.

In general, numerical Hessian updates are preferred in nonlinear programming. In the current version of our dynamic optimization library we implemented for SQP solvers dense BFGS (Fletcher, 1995) and SR1 (Nocedal and Wright, 1999; Tenny *et al.*, 2002) updates (both in modified versions for positive definiteness) for separate diagonal blocks of the Lagrangian Hessian. Those partitioned variable metric updates have proven to be very useful for solving DOCPs.

Finally, there is Powell's modified BFGS update operating on the whole matrix with the Gangster operator to avoid fill-in, but its application may lead to poor convergence of the SQP algorithm.

4.2.2. QP Solver. The computational complexity of algorithms for solving linear-quadratic optimization problems is mainly determined by the number of inequality constraints, besides the number of variables and of equality constraints.

Classical active-set algorithms identify the set of binding (active) inequality constraints at the solution point by step-wise index exchange operations. Unfortunately, they are non-polynomial, i.e., the worst-case number of performed steps increases faster than any polynomial with the number of inequality constraints. The most popular active-set strategy is the simplex algorithm for linear programming. Even though active-set strategies often perform well for many high-dimensional practical examples, there are other examples where their application is very inefficient.

Powell's algorithm implemented in our library uses the active-set strategy of Goldfarb and Idnani (Goldfarb and Idnani, 1983; Powell, 1985) with Fletcher's or Bertsekas's methods of active set initialization (Fletcher, 1987;

Bertsekas, 1982), but its effective application is limited to small-scale problems due to the number of active set steps necessary to find out the correct set of active inequality constraints.

The idea behind interior-point methods is to identify active inequality constraints numerically with the help of a barrier function. Their computational complexity has a polynomial bound (Karmarkar, 1984). In recent years, algorithms have been developed that show the excellent theoretical properties, confirmed also in practical applications.

Three different implementations of the interior-point algorithm are currently available for the SQP solver: `Mehrotra`, `Gondzio` and `Franke`. Mehrotra's primal-dual predictor-corrector method (Mehrotra, 1992) was implemented because of its good reputation in the literature for linear programming, e.g., (Wright, 1997). Recently we have added to Mehrotra's algorithm Terlaky's modification proposed in (Salahi *et al.*, 2005), which reasonably reduces the iteration complexity of the original algorithm. Gondzio's multiple centrality correctors solver (Gondzio, 1994) is a variant of Mehrotra's primal-dual interior-point method for convex QP, whose application for some problems may lead to a significant decrease in the total number of QP iterations and, as a consequence, to the reduction of the SQP solver runtime. Franke's QP solver is derived from several interior-point algorithms known from the literature, with most impact from Wright (Wright, 1993). Some modifications to this QP algorithm were introduced by R. Franke with the aim to improve its performance and robustness (Franke, 1994). According to our experience, the overall performance of Mehrotra's and Franke's QP solvers is about the same. However, for a specific problem one of them may perform significantly better than the other. To compare the performance of our specialized QP solvers with external, general-purpose quadratic programing solvers, we implemented interfaces to the solvers OOQP, BPMPD, MOSEK and LOQO. The most promising one is the OOQP solver, which for some problems was competitive with specialized QP solvers.

To explain the principal procedure, the optimization problem (8)–(10) is rewritten as

$$\min_s \left\{ \frac{1}{2} s^T H s + c^T s \;\mid\; As + b = 0,\; Cs + d \geq 0 \right\}. \tag{14}$$

H is supposed to be symmetric positive semidefinite and A must be of a full rank.

A barrier parameter $\mu > 0$ and a slack vector $w = Cs + d$ are introduced for the treatment of the inequality constraints. The objective function of (14) is augmented with a nonlinear barrier to

$$\min_s \frac{1}{2} s^T H s + c^T s - \mu \sum_{i=1}^{m_g} \ln w_i. \tag{15}$$

The solution of the quadratic subproblem is characterized by extended KKT conditions ($\Lambda = \mathrm{diag}(\lambda)$, $W = \mathrm{diag}(w)$, $e = (1 \ldots 1)^T$):

$$Hs + c - A^T p - C^T \lambda = 0, \tag{16}$$
$$As + b = 0, \tag{17}$$
$$w = Cs + d > 0, \tag{18}$$
$$\lambda > 0, \tag{19}$$
$$\Lambda W e - \mu e = 0, \tag{20}$$
$$\mu \to 0. \tag{21}$$

For each $\mu > 0$, Eqns. (16)–(20) have a unique solution (s, p, λ, w). For $\mu \to 0$, (20) reduces to the complementary condition $\lambda^T w = 0$; (16)–(20) then describe the KKT conditions of the original quadratic subproblem (14). The resulting vectors p and λ are used by the SQP solver as approximations of the Lagrangian multipliers.

Numerical solution of the nonlinear system (16)–(21) is obtained with an iterative Newton procedure. The initialization of a feasible starting point $(s^0, p^0, \lambda^0, w^0)$ and the control of the barrier parameter μ during the solution process are discussed in (Franke, 1994). The following linear equation system is passed to the matrix solver during each QP iteration j:

$$\begin{bmatrix} -H & A^T & C^T & 0 \\ A & 0 & 0 & 0 \\ C & 0 & 0 & -I \\ 0 & 0 & W_j & \Lambda_j \end{bmatrix} \begin{bmatrix} \delta s^j \\ \delta p^j \\ \delta \lambda^j \\ \delta w^j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Lambda_j W_j e - \mu_j e \end{bmatrix}. \tag{22}$$

The solution of this linear system is used to update the variable vectors:

$$\begin{aligned} &(s^{j+1}, p^{j+1}, \lambda^{j+1}, w^{j+1}) \\ &\quad := (s^j, p^j, \lambda^j, w^j) - \alpha_j (\delta s^j, \delta p^j, \delta \lambda^j, \delta w^j). \end{aligned} \tag{23}$$

The step length $\alpha_j \in [0, 1]$ is chosen maximal, provided that the new iterate $j + 1$ remains feasible.

A very interesting property of interior-point algorithms is that the failure of the complementary condition, i.e., $\lambda w = 0$ is continuously decreased over the iterations. In this way, one has a kind of measure for the distance of the current iterate from the optimum. This property may be exploited for the treatment of infeasible subproblem approximations.

4.2.3. Matrix Solver. Whereas the nonlinearities and the inequalities are treated by the SQP and the QP solver, respectively, the remaining computational complexity of the problem is passed to the matrix solver in the form of high-dimensional systems of linear equations.

The system (22) has to be solved once in each QP iteration. The coefficient matrix of (22) can be made symmetric by eliminating $\delta w^j = C \delta s^j$. This results in the

symmetric indefinite system:

$$\begin{bmatrix} -H & A^T & C^T \\ A & 0 & 0 \\ C & 0 & \Lambda_j^{-1} W_j \end{bmatrix} \begin{bmatrix} \delta s^j \\ \delta p^j \\ \delta \lambda^j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ W_j e - \mu_j \Lambda_j^{-1} e \end{bmatrix}. \quad (24)$$

We apply a diagonal scaling as proposed in (Wright, 1993) in order to improve the numerical stability.

In many cases it is advantageous to further reduce the system of equations prior to its factorization. This can be done by eliminating the inequality constraints from (24). This results in the reduced system:

$$\begin{bmatrix} -H - C^T W_j^{-1} \Lambda_j C & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \delta s^j \\ \delta p^j \end{bmatrix} = \begin{bmatrix} -C^T W_j^{-1} \Lambda_j r_3^j \\ 0 \end{bmatrix}, \quad (25)$$

$$\delta \lambda^j = W_j^{-1} \Lambda_j (r_3^j - C \delta s^j),$$
$$r_3^j = W_j e - \mu_j \Lambda_j^{-1} e.$$

The main computational effort of the interior-point method, as well as of the whole SQP algorithm, is the solution of the systems of linear equations described above. Here, the sparsity structure of the DOCP should be exploited. The main matrix solver for the SQP solver is based on the algorithm developed by E. Arnold and performs a block-wise elimination taking advantage of the stage-wise formulation of dynamic optimization problems. The algorithm is an extension of the Ricatti recursion for unconstrained linear-quadratic optimal control problems, see (Arnold *et al.*, 1994; Arnold and Puta, 1994). This specialized method for DOCPs outperforms clearly other matrix solvers with respect to computational time. An additional advantage of the method is that no dynamic allocation of memory is needed during the iterations if dense submatrices are used. The LQ-DOCP matrix solver uses dense linear algebra procedures of the NEWMAT or UBLAS matrix libraries and allows the calculation of the Newton step in $\mathcal{O}(N)$ operations (instead of $\mathcal{O}(N^3)$, as would be expected without taking into account the special structure of the linear system).

To compare the performance of the LQ-DOCP solver with external general-purpose sparse matrix solvers, we implemented the interfaces to direct solvers, MA27, MA47, WSMP, MUMPS, PARDISO, BLKFCLT, UMFPACK, and iterative solvers, GMRES, SYMMLQ, MINRES, SQMR. They are used to solve the sparse symmetric indefinite equation systems during iterations of the specialized QP solvers.

4.3. Interior-Point Methods for NLP Problems.

4.3.1. Introduction. Interior-point methods for nonlinear programming, also called barrier methods, rose from the need for effective solving of large-scale optimization problems. In particular, for NLP problems with large numbers of inequality constraints, these methods offer a serious alternative to active-set strategies. Within the last fifteen years researchers has led to a better understanding of the convergence of interior-point methods and has also developed effective computational algorithms with desirable global and local convergence properties.

The term *interior-point method* was used for the first time by Fiacco and McCormick in 1968 in the book (Fiacco and McCormick, 1968), for any algorithm that was designed for the calculation of a local minimum of an NLP problem by the solution of a determined sequence of unconstrained minimization problems. Such a definition evolved to the form, in which as of the IP method we think of any algorithm for solving a set of optimization problems associated with a decreasing value of the μ multiplier, to find local solutions lying in the interior of the feasibility set determined by nonlinear constraints of the NLP problem.

To allow convergence from "bad" starting points for interior-point methods in both trust region and line-search versions, researchers developed exact penalty merit functions that ensure progress toward the solution (Byrd *et al.*, 2000; Tits *et al.*, 2002). On the other hand, Fletcher and Leyffer (Fletcher and Leyffer, 2002; Fletcher *et al.*, 2006) proposed recently filter methods, as an alternative to merit functions which guarantee the global convergence for nonlinear programming algorithms. They are based on the idea of the approval of trial points generated by the optimization algorithm in the case when they improve the value of the objective function *or* improve the value of a constraint violation, instead of a combination of those two measures defined by a merit function.

More recently, this filter technique has been adapted to barrier methods. In (Ulbrich *et al.*, 2004), the authors consider a trust-region filter method, in which the consequent iterations of the solution are accepted on the basis of the norm of optimality conditions. Also, in (Benson *et al.*, 2001), the authors proposed several heuristics based on the concept of filter methods, for which the efficiency improvement was obtained as compared with their previous experience with merit functions. Finally, in (Wächter and Biegler, 2005), global convergence analysis of an interior-point algorithm with a filter line-search was provided.

Interior-point methods for NLP problems were implemented within many optimization solvers, such as LOQO (Vanderbei and Shanno, 1997), KNITRO (Byrd *et al.*, 1999; Waltz and Plantenga, 2006) or IPOPT (Wächter, 2002; Wächter and Biegler, 2006). In numerical

tests these solvers proved to be quite effective and robust for many large-scale NLP problems.

4.3.2. IPOPT Solver. In this section we describe a primal-dual interior-point algorithm with line-search minimization based on the filter method, used in the implementation of the IPOPT[2] solver that has recently been integrated into our library of dynamic optimization algorithms. The authors of the IPOPT assumed the following formulation of the original NLP problem:

$$\text{(P)} \qquad \min_{y\in\mathbb{R}^{m_y}} \{J(y) \mid h(y)=0,\ y\geq 0\}, \tag{26}$$

where the objective function $J\colon \mathbb{R}^{m_y} \mapsto \mathbb{R}$ and the equality constraints $h\colon \mathbb{R}^{m_y} \mapsto \mathbb{R}^{m_h}$ with $m_h < m_y$ are assumed to be twice continuously differentiable. NLP problems with general inequality constraints $g(y) \geq 0$ can be reformulated to the above form by introducing slack variables, i.e., s, where $g(y) - s = 0,\ s \geq 0$.

The barrier algorithm in the IPOPT solver is based on the replacement of the sign constraints on decision variables, $y \geq 0$, with an additional component in the objective function—the logarithmic barrier:

$$(\text{P}_\mu) \qquad \min_{y\in\mathbb{R}^{m_y}} \left\{ J_\mu(y) = J(y) - \mu \sum_{j=1}^{m_y} \ln(y^j) \mid h(y)=0 \right\}, \tag{27}$$

where $\mu > 0$ is the barrier parameter and y^j is the j-th element of the vector y. Since the objective function for Problem (P_μ) may attain arbitrarily large values when y^j reaches one of its bounds, the local solution $y_\star(\mu)$ to that problem is located in the interior of the set determined by the constraints $y_\star(\mu) > 0$. The scale of the barrier influence is determined by the size of the μ parameter and on certain assumptions, as $\mu \to 0$, the solution $y_\star(\mu)$ converges to a local solution $y_\star$ of the original problem (P). As a result, the algorithm to determine a solution to the original problem (P) is based on solving a sequence of barrier problems (P_μ) with decreasing values of the parameter μ_l, $\{\mu_l\} \to 0$.

The interior-point algorithm of the IPOPT solver finds a solution for primal-dual stationarity conditions for the problem (P_μ), formulated as the following nonlinear system of equations:

$$\nabla_y J(y) + \nabla_y h(y)\lambda - z = 0, \tag{28}$$

$$h(y) = 0, \tag{29}$$

$$YZ - \mu e_y = 0, \tag{30}$$

where Y and Z are diagonal matrices with elements y and z, respectively, e_y is the vector of ones of dimension m_y, $\lambda \in \mathbb{R}^{m_h}$ is the vector of Lagrange multipliers for equality constraints of the problem (P), and $z \in \mathbb{R}^{m_y}$ corresponds to the vector of Lagrange multipliers for the sign constraints of the problem (P), in the limit, as $\mu \to 0$. Note that the system of equalities (28)–(30) for $\mu = 0$, together with the additional condition $y, z \geq 0$, is equivalent to KKT optimality conditions for the original problem (P).

For the solution of the system of equalities (28)–(30), for a fixed value of the parameter μ, the IPOPT applies the iterative Newton method, based on the solution of the following system of linear equations:

$$\begin{bmatrix} W_k & \nabla_y h(y_k) & -I \\ \nabla_y h(y_k)^T & 0 & 0 \\ Z_k & 0 & Y_k \end{bmatrix} \begin{pmatrix} d_k^y \\ d_k^\lambda \\ d_k^z \end{pmatrix} = - \begin{pmatrix} \nabla_y J(y_k) + \nabla_y h(y_k)\lambda_k - z_k \\ h(y_k) \\ Y_k Z_k - \mu e_y \end{pmatrix}, \tag{31}$$

where W_k denotes the exact Hessian matrix for the Lagrange function of the original problem (P):

$$W_k = \nabla_{yy} J(y_k) + \sum_{i=1}^{m_h} \lambda_k^i \nabla_{yy} h^i(y_k), \tag{32}$$

or some approximation of it. The Lagrange function has the form

$$\mathcal{L}(y, \lambda, z) := J(y) + h(y)^T \lambda - z. \tag{33}$$

Here, the index k denotes the counter of inner iterations of Newton's method, the vector (y_k, λ_k, z_k) is the current iterate, and $(d_k^y, d_k^\lambda, d_k^z)$ is the obtained new search direction.

Instead of solving the nonsymmetric system of linear equalities (31) directly, the IPOPT solver obtains the equivalent solution by first solving the symmetric linear system of a smaller dimension:

$$\begin{bmatrix} W_k + \Sigma_k & \nabla_y h(y_k) \\ \nabla_y h(y_k)^T & 0 \end{bmatrix} \begin{pmatrix} d_k^y \\ \lambda_k^+ \end{pmatrix} = - \begin{pmatrix} \nabla_y J_\mu(y_k) \\ h(y_k) \end{pmatrix}, \tag{34}$$

where $\Sigma_k := Y_k^{-1} Z_k$. The equations for the system (34) are derived from those of the system (31) by eliminating the last block row. And then, the direction d_k^λ is computed from

$$d_k^\lambda = \lambda_k^+ - \lambda_k \tag{35}$$

and the direction d_k^z from

$$d_k^z = \mu Y_k^{-1} e_y - z_k - \Sigma_k d_k^y. \tag{36}$$

[2]An open source C++ version of IPOPT is available at `http://projects.coin-or.org/Ipopt`

After computing new search directions from (34)–(36), we calculate the next iterate as follows:

$$(y_{k+1}, \lambda_{k+1}, z_{k+1}) := (y_k, \lambda_k, z_k) + (\alpha_k d_k^y, \alpha_k d_k^\lambda, \alpha_k^z d_k^z), \quad (37)$$

where $\alpha, \alpha_k^z \in (0, 1]$ are the stepsizes. Note that for z variables it is allowed to take a different stepsize than for the other variables.

Since we know that the variables y and z are positive at an optimal solution of the barrier problem (P_μ), the IPOPT maintains this property for all iterates. As a result, the following rule of the step length selection is applied:

$$\bar{\alpha}_k := \max\{\alpha \in (0, 1] : \ y_k + \alpha d_k^y \geq (1 - \tau) y_k\}, \quad (38)$$

$$\bar{\alpha}_k^z := \max\{\alpha \in (0, 1] : \ z_k + \alpha d_k^z \geq (1 - \tau) z_k\}, \quad (39)$$

for the parameter $\tau \in (0, 1)$, usually close to 1 (e.g., $\tau = 0.995$). For the z variables the step length is chosen as $\alpha_k^z := \bar{\alpha}_k^z$, while the step length $\alpha_k \in (0, \bar{\alpha}_k]$ for the remaining variables is determined by a backtracking line-search procedure using a decreasing sequence of trial stepsizes, $\alpha_{k,l} = 2^{-l}\bar{\alpha}_k$, with $l = 0, 1, 2, \ldots$ — here a variant of Fletcher and Leyffer's filter method (Fletcher and Leyffer, 2002) is used, which guarantees global convergence of the interior-point algorithm to the solution of the problem (P_μ).

Filter minimization methods are based on the idea of two-criteria optimization in which, apart from minimizing the barrier objective function $J_\mu(y)$, we want to minimize the constraint violation $\theta(y) := \|h(y)\|$ in order to assure the convergence to a feasible point. In Fig. 1, a projection of the $\mathbb{R}^{m_y}$ space onto the $(\theta(y), J_\mu(y))$ halfplane was depicted. Every point from the original space of decision variables, such as the optimal solution $y_\star$ or the current iterate y_k, has its counterpart in this picture, e.g., $(\theta(y_k), J_\mu(y_k))$. The decision on accepting a trial point $y_k + \alpha_{k,l} d_k^y$ as the next solution iterate y_{k+1} depends on whether it guarantees a sufficient improvement of one of the two measures θ or J_μ as compared with their values at the point y_k. In the example of Fig. 1, the trial point "1" was not accepted because it worsens the values of both measures. Also the point "2" should be rejected, since it does not allow a sufficient decrease in the value of infeasibility (in the graph, sufficient decreases in the two measures are determined by two dashed lines whose intersection is closest to $(\theta(y_k), J_\mu(y_k))$). However, the trial point "3" should be accepted.

In the IPOPT solver the following safeguards have been added to this simple procedure of next iterate acceptance:

- In the case when the current iterate is (almost) feasible but not sufficiently optimal, the above condition of a sufficient decrease in one of two measures for y_k is replaced by the condition of a sufficient decrease in the barrier function value J_μ.
- In order to prevent cycling, the (θ, J_μ) pairs corresponding to previous iterate that create a certain envelope (in our example these are y_{l_1} and y_{l_2} iterates) are added to a *filter*; a trial point is only accepted if it guarantees a sufficient decrease in one of two measures relative to all those points. In our example, the trial point "4" should be rejected because it does not sufficiently improve the values of both measures θ and J_μ with respect to the y_{l_2} iterate.
- It may happen that there is no trial stepsize $\alpha_{k,l}$ that generates an acceptable point. After detecting such a situation, the algorithm switches to a *feasibility restoration phase* in which the minimization of infeasibility is carried out (ignoring the objective function) until either a new acceptable iterate is found or it is no longer possible to reduce the infeasibility, e.g., if the problem (P) is (locally) infeasible.

A formal description and analysis of the filter line-search procedure implemented in the IPOPT solver can be found in (Wächter and Biegler, 2005). In comparison with traditional line-search algorithms, such as a single merit function technique, the filter method is usually less conservative and makes it posible to take larger stepsizes. (In Fig. 1 for the method of an exact penalty function acceptable points will have to lie below a straight dotted line.) Moreover, the protection in the form of a restoration phase makes the filter algorithm resistant to unnecessary errors, such as those presented in (Wächter and Biegler, 2000).

The computationally most expensive part of the optimization algorithm implemented in the IPOPT solver (not including computations of the objective function, constraints and their derivatives) is the solution of the linear system of equations (34), which is most often of high order and has a sparse structure, e.g., for dynamic optimization problems it is very sparse. For its factorization and solution, the IPOPT uses external sparse direct linear solvers, such as MA27 (default option), MA57, WSMP, PARDISO and MUMPS.

4.3.3. SQP Methods vs. Nonlinear Interior-Point Methods. There is still a need for research on behaviour, effectiveness and robustness of sequential quadratic programming and nonlinear interior-point methods, see (Misc, 2003). At present, it seems that in terms of the effectiveness and robustness, nonlinear interior-point methods are better fitted for large-scale nonlinear optimization than SQP methods. And there is also a continuous development and competition between methods of both types—such a state will probably be lasting for the next few years.

The advantage of SQP methods over interior-point methods consists in their small number of required controlling parameters and in the possibility of using "good" starting points. On the other hand, SQP methods for degenerated problems have difficulties with the identifica-

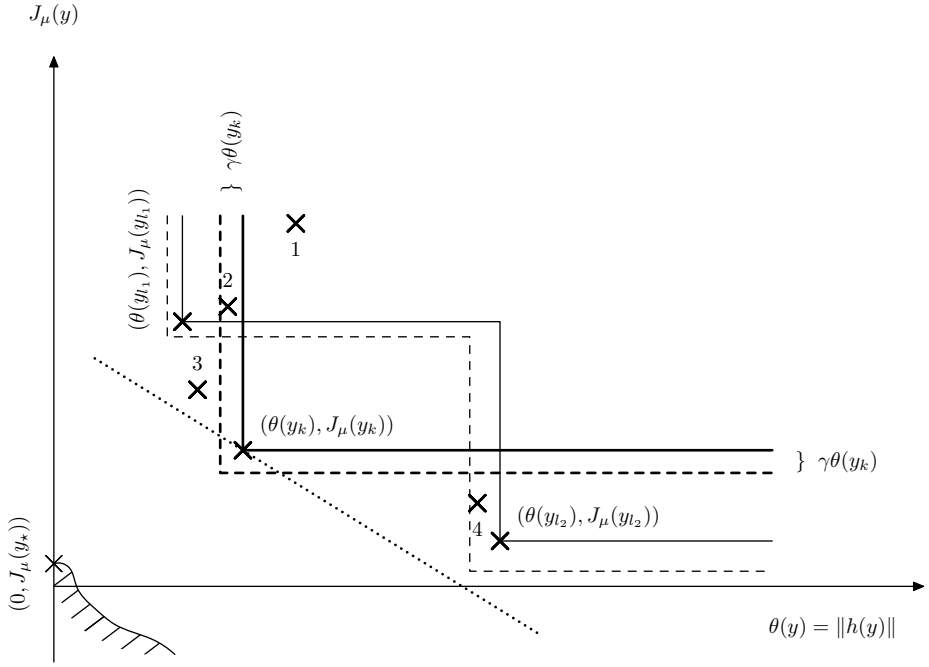

Fig. 1. Filter line-search method.

tion of the optimal active-set of constraints at the solution point.

On the other hand, nonlinear interior-point methods are a very efficient and effective optimization tool and, which is also important, they are not too sensitive to the degeneracy of the optimization problem. Their main flaw is dependency on controlling parameters, such as the barrier parameter, whose update requires complex heuristics. Moreover, their efficiency depends very much on the choice of the starting point.

In publications from the optimization domain one may find a lot of papers comparing the numerical efficiency of solvers based on SQP methods and nonlinear interior-point methods using the methodology of performance profiles, see (Dolan and Moré, 2002). It is worthwhile here to mention the articles (Benson *et al.*, 2002; Morales *et al.*, 2001), comparing the solvers SNOPT, filterSQP, LOQO and KNITRO, and the articles (Wächter, 2002; Wächter and Biegler, 2006), in which the authors compared the IPOPT solver with the LOQO and KNITRO solvers.

5. Dynamic Optimization of the Zambezi Reservoir System

The methods described above were applied to find out optimal release strategies for a system of reservoirs with hydroelectric power-stations on the Zambezi river in southern Africa (Arnold *et al.*, 1994). The dynamic optimization of that problem using the SQP method with the active-set strategy was done at the Institute of Control and Computation Engineering of the Warsaw University of Technology by E. Arnold as a part of the IIASA project *Water Resources* (Arnold *et al.*, 1994).

The structure of the reservoir system is given in Fig. 2. Its discrete-time model has a discretization step of one month and a control horizon ranging from one year up to 35 years. State equations are derived from volume balance equations of the reservoirs with the volumes of stored water as state variables. Control variables are the productive outflow (outflow through the turbines) and the spillage.

Balance equation ($i = 1, \ldots, 4, \quad k = 0, \ldots, N-1$):

$$V_i^{k+1} = V_i^k + Z_i^k + \sum_{j \in U_i} \left(Q_j^{k-\theta_j} + F_j^{k-\theta_j} \right) - Q_i^k - F_i^k - e_i^k f_{a,i}(V_i^k), \tag{40}$$

where the following notation is used:

V_i^k	–	reservoir volume; state variable,
Z_i^k	–	natural inflow (deterministic),
Q_i^k	–	productive outflow; control variable,
F_i^k	–	spillage; control variable,
U_i	–	upstream reservoirs,
θ_j	–	time delay,
$e_i^k f_{a,i}$	–	evaporation.

Since a deterministic control problem is considered, natural inflows are supposed to be known scenarios from historical data. Each of the balance equations contains a nonlinear evaporation term $e_i^k f_{a,i}(V_i^k)$. River dynamics between the reservoirs (time delay θ_j) are modelled by

linear difference equations with additional state variables. Thus, the four reservoir system has six state variables and seven control variables.

State and control bounds (or box constraints) arise from physical restrictions and additional demands, e.g., flood control and ecological demands, i.e.,

$$V^k_{\min,i} \le V^k_i \le V^k_{\max,i} \quad \text{(reservoir capacity)}, \tag{41}$$

$$Q^k_{\min,i} \le Q^k_i \le Q^k_{\max,i} \quad \text{(turbine flow bounds)}, \tag{42}$$

$$F^k_{\min,i} \le F^k_i \le F^k_{\max,i} \quad \text{(flood control etc.)}. \tag{43}$$

The main control goal is the maximization of the total electrical energy production. Therefore, the cost function includes a nonlinear mixed state-control term (the outflow multiplied by the state dependent water height level). Other terms of the cost function result from the desired small deviations in time of the energy production and the final water storage demands, respectively.

We have

$$\begin{aligned} J = \sum_{i=1}^{4} \Bigg\{ & \phi \left(V^N_i - V^N_{ref,i}\right)^2 - \nu_E f_{e,i}(V^N_i) \\ & + \sum_{k=0}^{N-1} \Big\{ - \alpha Q^k_i \left(f_{h,i}(V^k_i) + f_{h,i}(V^{k+1}_i)\right) \\ & + \psi \left(Q^k_i - Q^k_{\mathrm{ref},i}\right)^2 \Big\} \Bigg\}, \end{aligned} \tag{44}$$

where

$f_{h,i}$ – storage-water level relationship,
$V^N_{\mathrm{ref},i}$ – final reservoir volume reference value,
$Q^k_{\mathrm{ref},i}$ – reference trajectory for productive outflow,
$f_{e,i}$ – storage-potential electrical energy relationship,
α, ϕ, ν_E – weight parameters.

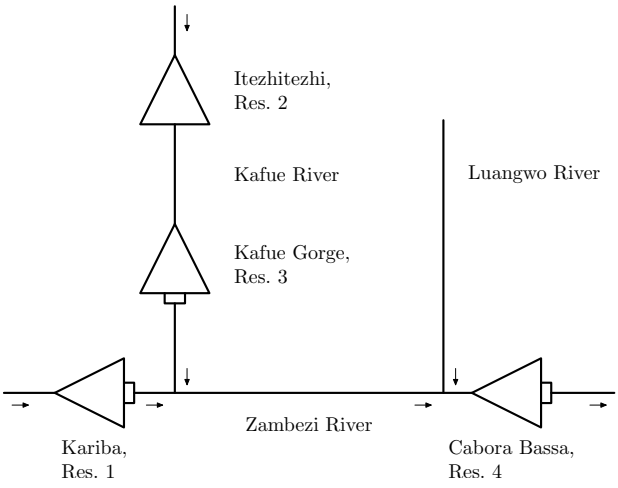

Fig. 2. Structure of the Zambezi river system.

This optimization problem fits into the class of optimal control problems defined in Section 3, if only the volume V^{k+1}_i in the water-level term of the cost function is replaced by the right-hand side of the balance equation.

In the calculations we used the following initial control trajectory ($i = 1, \ldots, 4, \;\; k = 0, \ldots, N-1$):

$$Q^k_i = Z^k_i + \sum_{j \in U_i} Z^k_j, \quad F^k_i = 0, \tag{45}$$

or a better one, determined from the reservoir balance equation, which assures faster convergence to the solution:

$$\begin{aligned} Q^0_i &= \begin{cases} Q^0_{\max,i} & \text{if } \bar{Z}_i > Q^0_{\max,i}, \\ \max(\bar{Z}_i, Q^0_{\min,i}) & \text{otherwise}, \end{cases} \\ F^0_i &= \begin{cases} \bar{Z}_i - Q^0_{\max,i} & \text{if } \bar{Z}_i > Q^0_{\max,i}, \\ 0 & \text{otherwise}, \end{cases} \\ Q^k_i &= Q^0_i, \quad k = 1, \ldots, N-1, \\ F^k_i &= F^0_i, \quad k = 1, \ldots, N-1, \end{aligned} \tag{46}$$

where

$$\begin{aligned} \bar{Z}_i = \frac{1}{N} & \left(\sum_{k=0}^{N-1} Z^k_i - (V^N_{\mathrm{ref},i} - V^0_i) \right) \\ & + \sum_{j \in U_i} \left(Q^0_j + F^0_j\right). \end{aligned} \tag{47}$$

The `Zambezi` problem was formulated as six different computational examples for which data were taken from the following sources:

zambezi8, zambezi9, zambezi10, zambezi11, zambezi come from the CUTEr[3] collection of optimization test problems and correspond to the problems `ZAMB2-8.SIF`, `ZAMB2-9.SIF`, `ZAMB2-10.SIF`, `ZAMB2-11.SIF`, `ZAMB2.SIF` from the small test problem set `mastsif_small.tar.gz`. The **zambezi10** example was discussed in (Arnold *et al.*, 1994), whereas in (Arnold and Puta, 1994) the results for the **zambezi9** example were presented.

zambezi12 is the modification of the **zambezi10** example described in (Franke and Arnold, 1997). In that problem the influence of the reference control trajectory $Q^k_{\mathrm{ref},i}, k = 0, \ldots, N-1$ on the cost function was omitted, whereby the bounds on state variables and controls are of great importance for the problem solution.

[3] `http://cuter.rl.ac.uk/cuter-www/`

The historical data about natural inflows to reservoirs and all values of problem parameters were taken from the appropriate CUTEr test problem files.

The dynamic SQP solver and the IPOPT solver performed satisfactorily with high accuracy for all variations of the Zambezi problem with varying time horizons, varying inflow scenarios and cost function parameters. Examples of optimal trajectories for the **zambezi12** example with $N = 24$ are shown in Fig. 3. As be can seen, the optimal control trajectories for Q_i variables are most often located on bounds.

Examplary calculations reported in this section were performed on a computer with an AMD Athlon 64 X2 2.2 GHz processor, with 2 GB RAM, running Linux 2.6. All examples were treated with default settings of solvers and we used the initial control trajectory given by Eqns. (46) and (47). We compared the SQP solver using Franke's interior-point (SQP-IP) algorithm with two alternative solvers: the SQP solver using an active-set strategy (SQP-AS) and the IPOPT optimizer. We wanted to show how the discussed algorithms behaved for dynamic optimization problems with increasing dimensions.

The results of optimization for the **zambezi12** example comparing the three different solvers are summarized in Table 1. This table shows:

1. the number of stages, variables, equality constraints (including fixed variables), and inequality constraints (including bounds), respectively,
2. the number of main iterations,
3. the cumulative number of minor iterations (i.e., QP iterations for the SQP method),
4. the computational time (in seconds) for two implementations of dense linear algebra operations in solvers: one using the NEWMAT (N) matrix library and the other using the UBLAS (U) matrix library. For the IPOPT solver, only the run time for the UBLAS implementation was shown because both of the matrix libraries are used only for problem formulation and the computation of derivatives, while the computational code of the IPOPT uses native C++ matrix classes.

The conclusions from that comparison of solvers are as follows:

- Both of the SQP solvers required a similar, rather small number of main iterations, whereas the number of iterations of the IPOPT solver is relatively high.
- For small problems (up to about 1000 variables) the SQP-AS solver was more effective than the SQP-IP solver.
- For large problems the disadvantage of the SQP-AS solver is long computational time needed to identify the set of active inequality constraints.
- The SQP-IP solver shows a very moderate increase in the computational time. The effort is high for small problems, due to the interior-point method, but with an increasing problem dimension (above 1000 variables) this initial burden is more than compensated by an better computational complexity of the algorithm.
- By using faster dense linear algebra operation provided by the UBLAS library (and ATLAS procedures called by UBLAS), we doubled the speedup for the SQP-IP solver and tripled it for the SQP-AS solver compared with less efficient implementation of matrix operations offered by the NEWMAT library.
- For small problems the performance of the general-purpose NLP solver (IPOPT) is noticeably worse than that of the SQP solvers specialized for dynamic optimization, but while increasing the problem dimension it becomes comparable with the performance of the SQP-AS solver.

6. Benchmarking SQP and Nonlinear Interior–Point Methods

In order to compare the performance of different versions of given solver for dynamic optimization or to compare a few different solvers running on the same set of test problems, the methodology of performance profiles proposed in (Dolan and Moré, 2002) was used. This approach is based on the computation of the probability estimate that an algorithm performs within a multiple of the runtime or iteration count (or any other metric) of the best algorithm.

Assume that we compare the performance of n_s solvers on a testing set consisting of n_p problems. For the case of using the run time of the algorithm as the metric in the calculated performance profile, we introduce the following notation:

$$t_{p,s} = \text{run time required to solve problem } p \text{ by solver } s. \tag{48}$$

Then the *performance ratio* of the solver s on the problem p is defined by

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} \,:\, 1 \le s \le n_s\}}. \tag{49}$$

Moreover, for test problems for which we cannot find a solution by a given solver in finite time, we assume $r_{p,s} = r_M$, where r_M is a sufficiently large number. For successfully solved problems we have $r_{p,s} \le r_M$.

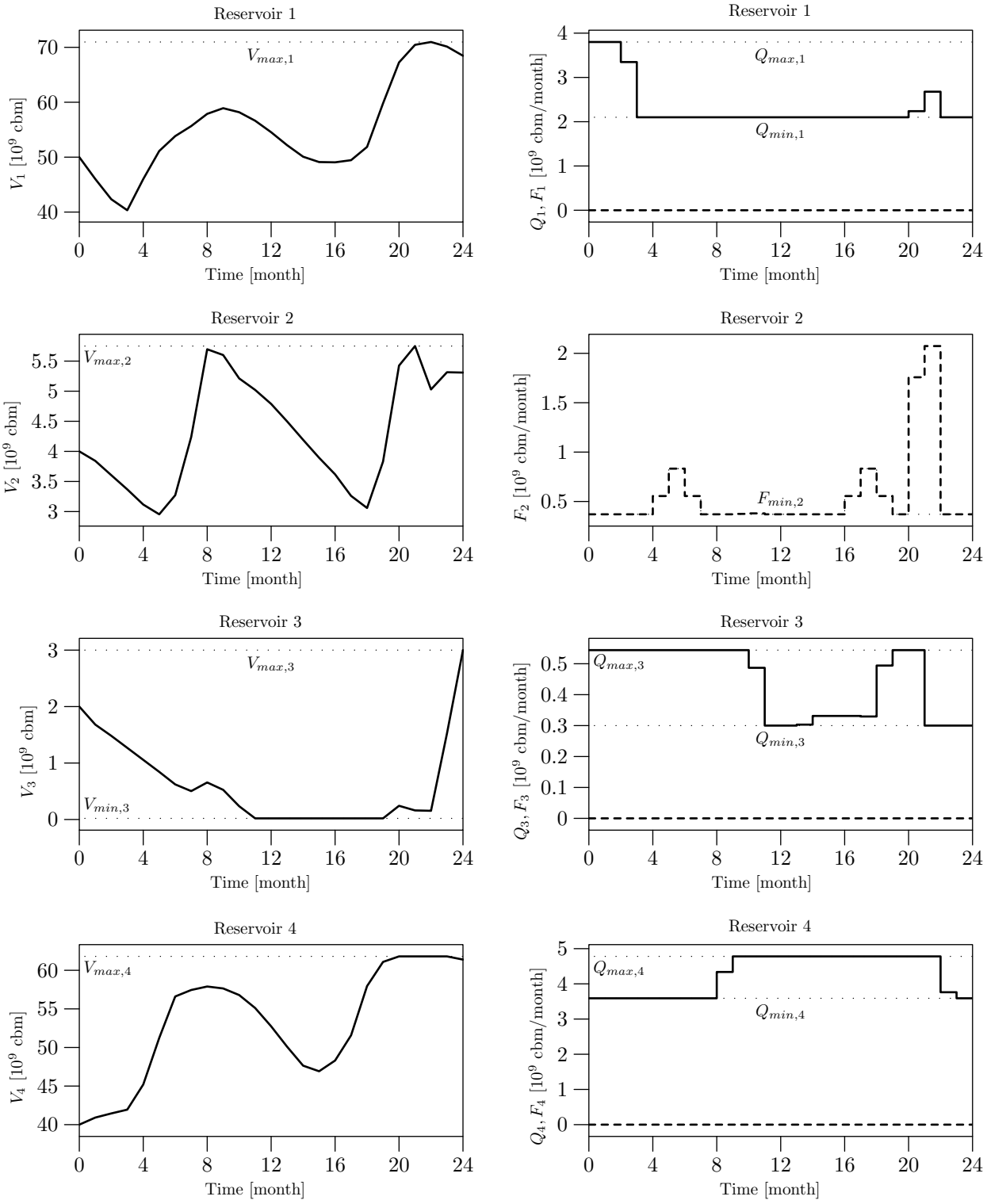

Fig. 3. Optimal state and control trajectories for the **zambezi12** problem with a 24-month control horizon (1945–1946). The state variables V_i and controls Q_i are drawn with the solid line, the controls F_i with dashed line, and bounds with the dotted line.

Table 1. Computational results for the **zambezi12** example with the horizon increasing from 12 to 252 months. The number of main iterations, cumulative numbers of minor iterations, and CPU times (in seconds) are listed for three different solvers. The times are taken using a PC with an AMD Athlon 64 X2 processor with a 2.2 GHz clock.

Problem size				SQP-IP				SQP-AS				IPOPT	
N	n	m_e	m	Iter	QP-Iter	Time (N)	Time (U)	Iter	QP-Iter	Time (N)	Time (U)	Iter	Time (U)
12	162	78	272	3	65	0.11	0.06	7	87	0.07	0.03	19	0.15
24	318	150	536	4	60	0.21	0.11	3	132	0.17	0.07	23	0.35
36	474	222	800	10	232	1.45	0.58	10	282	0.52	0.22	31	0.71
48	630	294	1064	8	161	1.07	0.52	7	283	0.66	0.27	33	1.00
60	786	366	1328	7	233	1.90	0.92	7	387	1.09	0.43	34	1.30
72	942	438	1592	8	166	1.67	0.83	8	446	1.48	0.58	35	1.61
84	1098	510	1856	9	185	2.19	1.11	10	735	2.84	1.10	36	1.95
96	1254	582	2120	11	207	2.88	1.46	11	795	3.56	1.40	41	2.57
108	1410	654	2384	9	201	3.17	1.61	9	847	4.53	1.73	47	3.38
120	1566	726	2648	10	206	3.69	1.86	9	835	5.22	1.97	43	3.41
132	1722	798	2912	11	360	6.83	3.47	10	1047	7.49	2.83	47	4.26
144	1878	870	3176	11	387	7.74	4.26	9	1131	9.06	3.35	44	4.41
156	2034	942	3440	11	341	7.79	4.02	9	1256	11.19	4.14	45	4.82
168	2190	1014	3704	11	305	7.73	3.99	9	1329	12.65	4.63	51	6.17
180	2346	1086	3968	11	350	9.38	4.88	9	1372	13.94	5.05	50	6.59
192	2502	1158	4232	12	365	10.52	5.48	8	1470	15.97	5.83	48	6.98
204	2658	1230	4496	10	348	12.42	5.74	9	1439	16.59	6.15	48	7.69
216	2814	1302	4760	11	355	12.16	6.24	9	1686	21.71	7.99	47	8.25
228	2970	1374	5024	21	423	15.28	8.12	17	1914	26.39	10.01	51	9.96
240	3126	1446	5288	12	470	17.09	8.91	10	2032	29.69	11.19	50	10.49
252	3282	1518	5552	14	424	16.18	8.61	12	2033	30.49	11.46	50	11.43

In order to evaluate the performance of an algorithm on the entire set of problems, one can use the quantity

$$\phi_s(\tau) = \frac{1}{n_p}\text{card}\{p : \ 1 \le p \le n_p, \quad \log_2(r_{p,s}) \le \tau\}. \tag{50}$$

The function $\phi_s : \mathbb{R} \mapsto [0,1]$ is called the logarithmic *performance profile* of the solver s and represents the cumulative distribution function of the performance ratio $r_{p,s}$. It is a nondecreasing, piecewise constant function, right-continuous at each breakpoint. The value of $\phi_s(1)$ is the probability that the solver wins over the rest of the solvers. So, if we are interested only in the number of winnings, we have to compare only the values of $\phi_s(1)$ for all solvers.

Defining the performance profiles for larger values of τ, we assume that $r_{p,s} \in [1, r_M]$ and that $r_{p,s} = r_M$ only when it is not possible to solve the problem p by the solver s. As a result of this convention, $\phi_s(r_M) = 1$, and we can define the probability that the solver solves a problem as follows:

$$\phi_s(r_M^-) \equiv \lim_{\tau \to r_M^-} \phi_s(\tau). \tag{51}$$

Thus, if we are interested only in solvers guaranteeing a high probability of success, we should choose solvers with the largest value of $\phi_s(r_M^-)$.

The so-defined performance profiles, apart from the run time criteria, may be used also for comparing the number of iterations and the number of objective function evaluations by benchmarked solvers.

It turns out that performance profiles eliminate the influence of a small number of problems on the process of benchmarking optimization methods and the sensitivity of results associated with the ranking of solvers. They provide means of visualizing the expected performance difference among many solvers. Using them we avoid an arbitrary choice of parameters used for comparisons and we must not discard solver failures from the performance data. If for the comparisons of solvers we use a suitably

large and representative set of test problems, then solvers with large probability $\phi_s(\tau)$ are to be preferred.

The performance profiles were used to compare different options of dynamic optimization solvers based on the SQP method with an interior-point QP solver. As the test problems for benchmarking, we use different versions of the `Zambezi` problem with the varying number of stages of the control horizon N: **zambezi** ($N = 12 : 12 : 420$), **zambezi8** ($N = 12 : 12 : 216$), **zambezi9** ($N = 12 : 12 : 72$), **zambezi10** ($N = 12 : 12 : 72$), **zambezi11** ($N = 12 : 12 : 432$) and **zambezi12** ($N = 12 : 12 : 252$). For every dynamic optimization problem from a total of 122 testing problems, two initial control trajectories were used (the first described by (45), and the second given by (46) and (47), for which the SQP algorithm has much better convergence to the solution). Thus, finally the testing set counted 244 problems. All computations were carried out in the case of the use of the UBLAS matrix library. The following options of the dynamic SQP solver were compared:

- Various matrix solvers in the interior-point QP algorithm used for the solution of the sparse symmetric indefinite system of linear equations (full or reduced). They were: the LQ-DOCP method, specialized for dynamic optimization problems, and external sparse direct matrix solvers: MA27, MA47, WSMP (in the full and reduced versions), PARDISO, BLKFCLT, MUMPS and UMFPACK. They were not compared with external dense direct solvers and iterative solvers (SYMMLQ, MINRES, SQMR) because of their low numerical efficiency for larger test problems.

- Various methods of the Lagrangian Hessian update: the block BFGS method with the Powell modification, the block SR1 method, the exact Hessian computation by automatic differentiation, the Gerschgorin method and the Gangster method.

- Methods for quadratic programming: specialized for dynamic optimization problems interior-points methods of Franke, Mehrotra and Gondzio and methods based on the use of external QP solvers: OOQP (in Mehrotra's and Gondzio's versions), MOSEK, BPMPD and LOQO.

On the basis of the data gathered during the optimization by the dynamic SQP solver and by the usage of the Perl script `perf.pl` (available from the web pages of the COPS[4] project), the performance profiles were generated for benchmarked options of the SQP solver. The metrics used were the computation time, the required number of major SQP iterations and the total number of QP iterations. The obtained performance profiles are shown in Figs. 4–6, respectively. In the case of the lack of convergence of the SQP algorithm, the stopping criterion was the maximal allowed number of iterations (500).

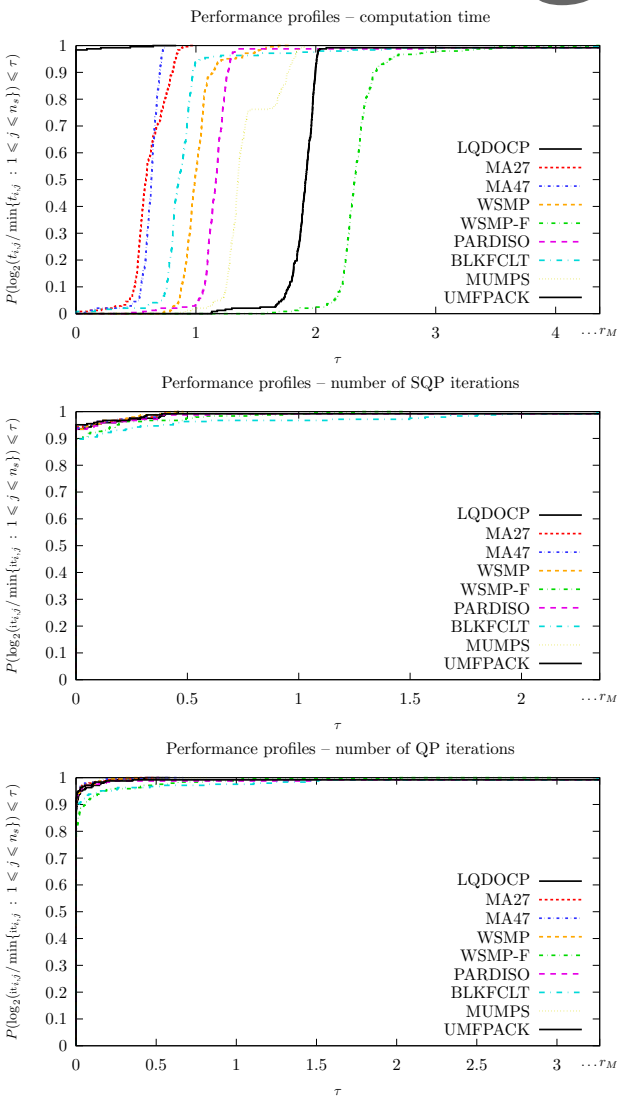

Fig. 4. Performance profiles comparing the computational time, number of major iterations and total number of QP-iterations for the SQP solver in the case of using different matrix solvers; the `Zambezi` testing set was considered.

From Fig. 4 it is clear that among the matrix solvers the specialized LQ-DOCP solver with the greatest probability (equal to 0.98) fulfils the criteria to be the best matrix solver (as regards the computation time) for an interior-point method of the dynamic SQP solver. The external sparse matrix solvers are far more worse and there is no need to apply them during the optimization. Among them, two procedures from the HSL library (MA27 and MA47) were the best – they had very close shapes of the performance profiles. Next positions were taken by BLKFCLT, WSMP, PARDISO and MUMPS solvers—all designed for symmetric matrices. The slowest matrix solver was UMFPACK, designed for general matrices, and the

[4] `http://www-unix.mcs.anl.gov/~more/cops/`

WSMP solver, operating on the full linear system. In the case of external solvers the time needed to convert matrices of the QP problem (stored in the own format managed by the class `SparseBlockMatrix`) into formats used by the matrix solvers was added to the total computation time. This conversion is a time consuming operation, but it is needed at every iteration of the interior-point method. Almost all matrix solvers were effective in 100%—only in the case of the PARDISO, BLKFCLT and UMFPACK solvers, the isolated cases of the lack of the convergence of the SQP algorithm were observed. The performance profiles using the number of QP and SQP iterations as metrics for all the matrix solvers are almost identical, which shows that for the majority of test problems we obtained very close or simply identical courses of the SQP algorithm iterations and values of the controlling parameters.

From the comparison of update methods of the Lagrangian Hessian for the SQP solver (presented in Fig. 5) it follows that the most effective method is exact Hessian computation by automatic differentiation – in all three metrics (the computation time $P = 0.67$, the number of major iterations $P = 0.82$, the number of QP-iterations $P = 0.82$) it was the winner of this performance profile benchmark. Unfortunately, it was also the least reliable in our computations (the probability of solving all problems is equal to $P = 0.84$), which results from the fact that the exact Hessian of the Lagrange function may be not positive definite and the resulting QP problem is nonconvex. From methods of the Hessian update with convergence guarantee equal to 100% the fastest are block methods of SR1 ($P = 0.2$) and BFGS ($P = 0.11$), and, in addition, the former has by far a better performance profile. The Gerschgorin and Gangster methods are slower than the other methods. Moreover, for the latter, solutions for 14% of test problems were not obtained. Therefore, the the recommended update method of the Lagrangian Hessian for the SQP solver is the block SR1 method, being the most effective, and, at the same time, robust. Let us notice that the computation of the second derivatives for the functions of the dynamic optimization problem by automatic differentiation procedures of the effective ADOL-C library allowed very efficient implementation of the exact Hessian computation.

The comparison of quadratic programming methods for the SQP algorithm (see Fig. 6) definitely proved the advantage of the Mehrotra interior-point method specialized for dynamic optimization problems, both in terms of the computation time ($P = 0.85$) and the number of QP-iterations ($P = 0.96$). The second, in terms of the computation time ($P = 0.09$), was the specialized Franke method, while other methods are definitely slower and have rather similar performance profiles for the run time metric. Here, an exception is the BPMPD solver, which in our tests was least effective, not allowing solving 25%

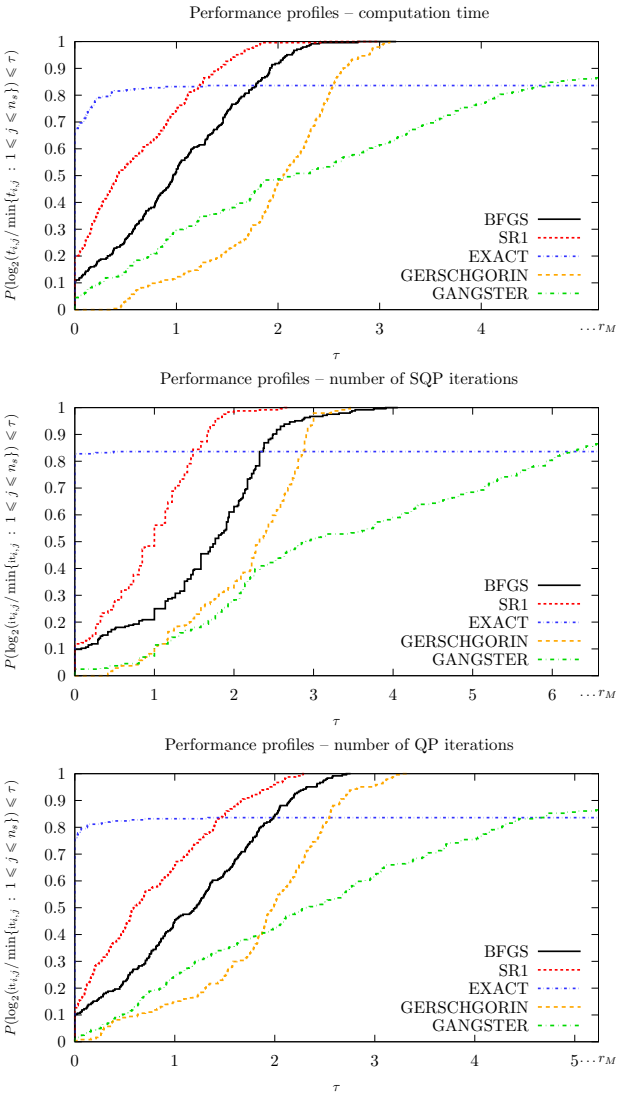

Fig. 5. Performance profiles comparing the computation time, the number of major iterations and the total number of QP-iterations for the SQP solver in the case of using different update methods for the Hessian of Lagrange function; the `Zambezi` testing set was considered.

of test problems. Moreover, its computation time was extended by the needed recording of iteration data to external files, which certainly was an important share in the total runtime of the SQP solver for smaller test problems. In the case of using the OOQP solver, the total number of QP-iterations inside the SQP algorithm could be lower if that solver had the possibility of a warm start. Generally speaking, the interior-point methods of Franke, Mehrotra and Gondzio, specialized for dynamic optimization problems, came out much better in terms of both the computation time and the total number of QP-iterations required than the methods based on using external QP solvers, such as OOQP (in the Mehrotra and Gondzio versions), MOSEK, BPMPD and LOQO. This can be explained by the fact that the specialized methods use the LQ-DOCP

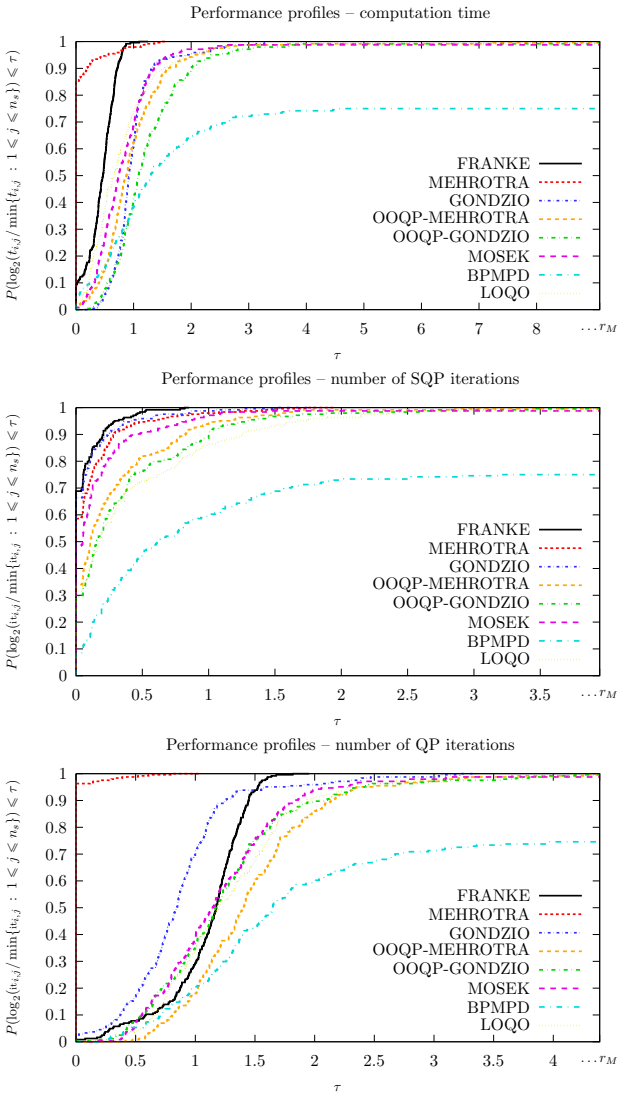

Fig. 6. Performance profiles comparing the computation time, the number of major iterations and the total number of QP-iterations for the SQP solver in the case of using different quadratic programming solvers; the `Zambezi` testing set was considered.

matrix solver for dynamic optimization problems, and the general QP-solvers use for that purpose general (i.e., slower) sparse matrix solvers (e.g., the OOQP solver uses the MA27 Fortran procedure form the HSL library). Let us remind here, the fact that the run time of the QP solver is dominated by the computational effort needed for solving a sequence of linear systems with sparse symmetrical indefinite matrices. As for the required number of major SQP iterations, the performance profiles for QP solvers are similar. In addition, some advantage is attributed to the specialized Franke interior-point solver.

The testing set of 244 `Zambezi` problems was also used to compare the efficiency of different versions of the dynamic SQP solver and the method based on the solution of the dynamic optimization problem by the application of a general nonlinear programming solver—we chose the nonlinear interior-point optimizer IPOPT. For comparison, we used the following methods:

- the Powell SQP solver with an interior-point method for QP,
- the Schittkowski SQP solver with an interior-point method for QP,
- the Powell SQP solver with active-set strategy for QP,
- the old version (2.2.1) of the IPOPT solver (implemented in Fortran),
- the new version (3.2.2) of the IPOPT solver (implemented in C++).

For the discussed solvers, the performance profiles were generated using the following metrics: computation time, the required number of algorithm iterations and the number of objective function evaluations. They are displayed in Fig. 7. All compared solvers solved all test problems. In terms of the computation time, the fastest solver turned out to be the SQP solver with an active-set strategy ($P = 0.39$) and the C++ version of the IPOPT solver ($P = 0.37$). For the runtime metric the performance profiles for the SQP solvers of Powell ($P = 0.18$) and Schittkowski ($P = 0.08$) and the old version of the IPOPT solver are worse than others. Let us notice that the plots of performance profiles for the Powell and Schittkowski SQP solvers are very close to each other. In terms of the number of iterations and objective function evaluations, the SQP solver with an active-set strategy was also the best one . The second place was taken by the Powell SQP solver with the interior-point method, and a bit worse from it was the Schittkowski SQP solver. However, both of the versions of the IPOPT solver required definitely large numbers of iterations and objective function evaluations. Their performance profiles for these measures were identical. To sum up, the computational performance of the SQP methods, specialized for dynamic optimization problems and using only the first derivatives, is quite similar to the performance of a general nonlinear interior-point method implemented in the IPOPT solver. The good results of the IPOPT solver in this benchmark are largely due to using second derivatives of problem functions, effectively calculated owing to automatic differentiation techniques and considerably accelerating the convergence of solver iterations. Moreover, it is worth noting the performance improvement after the reimplementation of the IPOPT solver from Fortran to the C++ language.

It is interesting to study performance profiles for larger test problems, as shown in Fig. 8. From the whole `Zambezi` testing set we chose for comparison problems

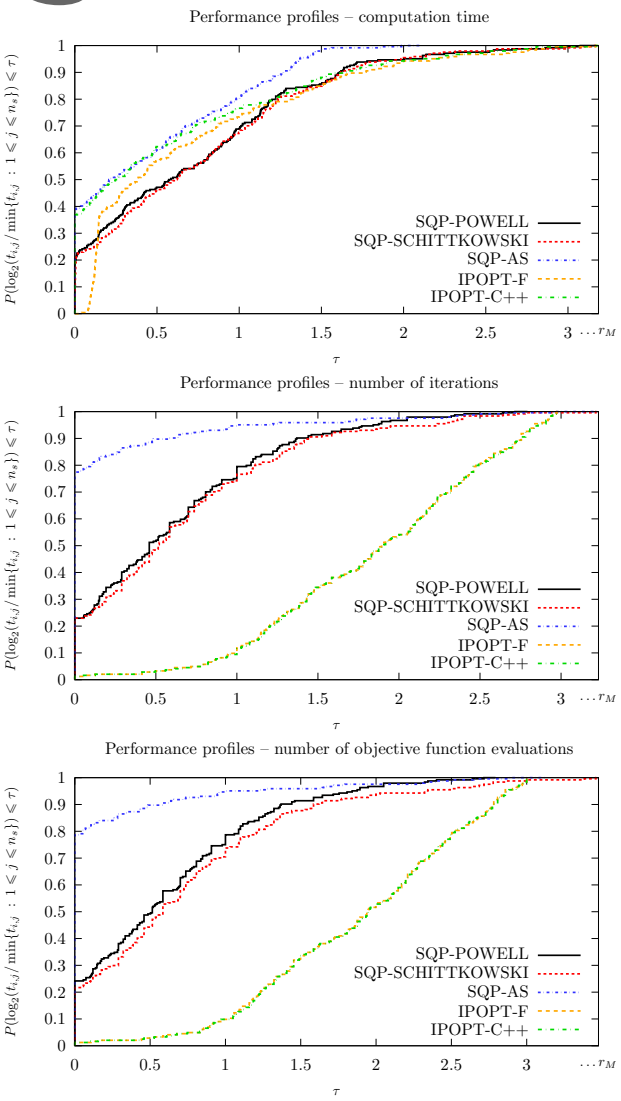

Fig. 7. Performance profiles comparing the computational time, the number of iterations and the number of objective function evaluations for different versions of SQP and IPOPT solvers; the `Zambezi` testing set was considered.

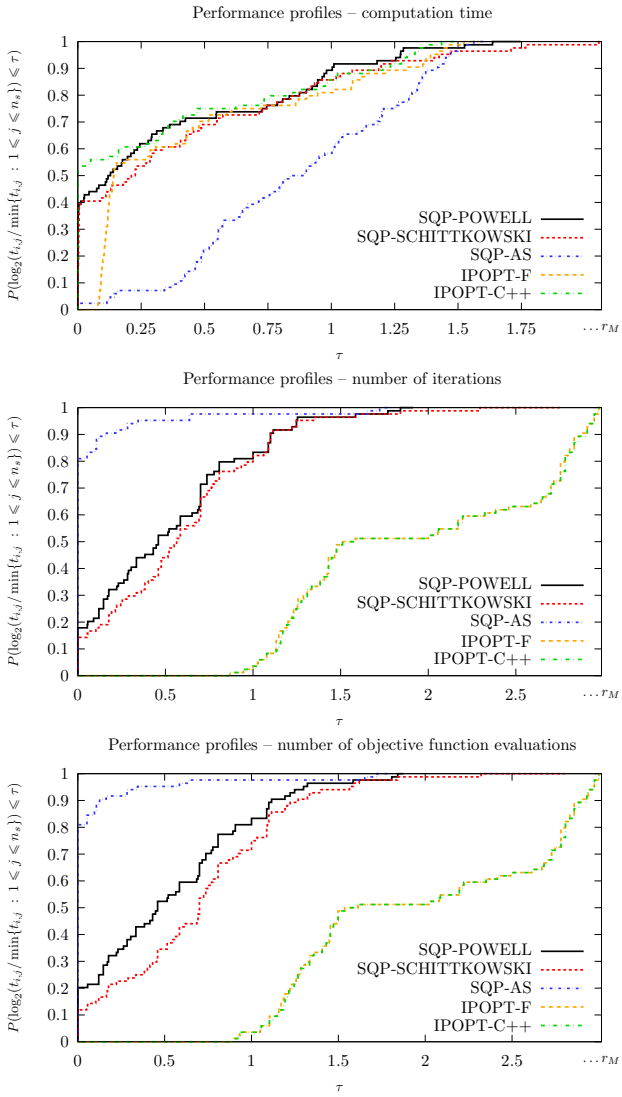

Fig. 8. Performance profiles comparing the computation time, number of iterations and objective function evaluations number for different versions of SQP and IPOPT solvers; the `Zambezi` testing set consisting with problems with $N \geq 216$ was considered.

with the number of stages of the control horizon $N \geq 216$, which gave 84 test problems. For large problems, both of the SQP solvers with an interior-point method and the IPOPT solvers in terms of the computation time had quite similar performance profiles, while the SQP solver with an active-set strategy was definitely the weakest one ($P = 0.02$) in that benchmark. It is a result of the fact that for problems with a larger number of decision variables the required number of active set changes in the quadratic programming subproblem increases considerably, which significantly extends the total runtime of the SQP method. For larger test problems, the C++ version of the IPOPT solver was the best—for 54% of test problems it was the fastest. The worst solver (SQP with an active-set strategy) for about 58% of large problems was no more than twice worse than the best solver for every test problem. For large problems, the performance profiles for a number of iterations and objective function evaluations are similar to the solvers on the whole testing set.

7. Conclusions

The main achievement of the presented work was the implementation of an object-oriented numerical library programmed in the C++ language; this allows easy formulation and effective solution of a broad range of dynamic optimization problems. The library was released under the name OLADO (**O**bject **L**ibrary of **A**lgorithms for **D**ynamic **O**ptimization) and its C++ source code is freely available upon request from the authors.

The experience gathered during designing and programming our library allows us to state that adopting the object-oriented methodology for the construction of numerical software was a good decision. By using object-oriented programming techniques, in particular inheritance and polymorphism, we obtained flexible software that could be used for a wide spectrum of numerical experiments: from investigating general performance of different algorithms of dynamic optimization to the analysis of the influence of various parameters in the problem formulation and in the numerical algorithm on the optimization results.

The class hierarchy we created is extensible—for an experienced C++ programmer it is relatively simple to implement and test new algorithms of dynamic optimization. Our library also provides facilities for effective computation of exact values of the first and second derivatives of the functions in dynamic optimization problem formulations—we used automatic differentiation utilities provided by the ADOL-C library. For linear algebra computations a potential user of our library can use vector and matrix classes of the NEWMAT or UBLAS libraries with our additional extensions.

The current version of the library contains several methods of dynamic optimization without constraints, mainly classical ones, based on computation of the reduced gradient of the functional with respect to control, and also two methods based on the dynamic programming algorithm. For problems with simple constraints on control we have the method of projection with different versions of computing the improvement direction. Constrained DOCPs, after their transformation into large NLP problems, are solved efficiently by various SQP methods (especially if a special structure of the dynamic optimization problem is exploited during the solution of a sequence of QP subproblems) or general-purpose NLP solvers, such as IPOPT, LOQO, and KNITRO.

Our main contribution when developing the dynamic optimization library is the implementation of advanced numerical algorithms, based on available software tools. The library solvers show very interesting performance for the solution of discrete-time optimal control problems, especially those caused by the application of specialized SQP-type methods with interior-point QP solvers, and also using the general large-scale nonlinear interior-point solver IPOPT. The efficiency and robustness of those two main library solvers for constrained problems was investigated using the performance profile methodology on a large testing set of dynamic optimization problems.

References

Arnold E. and Puta H. (1994): *An SQP-type solution method for constrained discrete-time optimal control problems*. In: Computational Optimal Control (R. Bulirsch and D. Kraft, Eds.), Birkhäuser Verlag, Basel, Switzerland, pp. 127–136.

Arnold E., Tatjewski P. and Wołochowicz P. (1994): *Two methods for large-scale nonlinear optimization and their comparison on a case study of hydropower optimization.* Journal of Optimization Theory and Applications, Vol. 81, No. 2, pp. 221–248.

Benson H. Y., Shanno D. F. and Vanderbei R. J. (2001): *Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions*. Technical Report ORFE-00-06, Operations Research and Financial Engineering, Princeton University. Available at `http://www.princeton.edu/~rvdb/tex/loqo4/loqo4_4.pdf`.

Benson H. Y., Shanno D. F. and Vanderbei R. J. (2002): *A comparative study of large-scale nonlinear optimization algorithms.* Technical Report ORFE-01-04, Operations Research and Financial Engineering, Princeton University. Available at `http://www.princeton.edu/~rvdb/tex/loqo5/loqo5_5.pdf`.

Bertsekas D. P. (1982): *Projected Newton methods for optimization problems with simple constraints.* SIAM Journal on Control and Optimization, Vol. 20, No. 2, pp. 221–246.

Błaszczyk J., Karbowski A. and Malinowski K. (2002a): *Object library of algorithms for unconstrained dynamic optimization problems.* Proceedings of the 14-th National Conference on *Automatic Control (KKA)*, Vol. I, Zielona Góra, Poland, pp. 451–456.

Błaszczyk J., Karbowski A. and Malinowski K. (2002b): *Object library of algorithms for dynamic optimization problems without constraints or with simple bounds on control.* Proceedings of the 8th IEEE International Conference on *Methods and Models in Automation and Robotics*, Vol. 1, Szczecin, Poland, pp. 257–262.

Błaszczyk J., Karbowski A. and Malinowski K. (2003): *Object library of algorithms for dynamic optimization problems with general constraints.* Proceedings of the 9th IEEE International Conference on *Methods and Models in Automation and Robotics*, Vol. 1, Międzyzdroje, Poland, pp. 271–276.

Bryson A. E. (1998): *Dynamic Optimization*. Menlo Park CA: Addison-Wesley, p. 550.

Byrd R. H., Hribar M. E. and J. Nocedal (1999): *An interior point algorithm for large scale nonlinear programming.* SIAM Journal on Optimization, Vol. 9, No. 4, pp. 877–900.

Byrd R. H., Gilbert J. Ch. and Nocedal J. (2000): *A trust region method based on interior point techniques for nonlinear programming.* Mathematical Programming, Vol. 89, pp. 149–185.

Chamberlain R. M., Powell M. J. D., Lemarechal C. and Pedersen H. C. (1982): *The watchdog technique for forcing convergence in algorithms for constrained optimization.* Mathematical Programming Study, Vol. 16, pp. 1–17.

Dolan E. D. and Moré J. J. (2002): *Benchmarking optimization software with performance profiles.* Mathematical Programming, Vol. 91, No. 2, pp. 201–213.

Fiacco A. V. and McCormick G. P. (1968): *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley and Sons, New York/London.

Findeisen W., Szymanowski J. and Wierzbicki A. (1980): *Theory and Computational Methods of Optimization*. Polish Scientific Publishers, Warsaw (in Polish).

Fletcher R. (1987): *Practical Methods of Optimization*. John Wiley and Sons, New York, NY, USA.

Fletcher R. (1995): *An optimal positive definite update for sparse Hessian matrices.* SIAM Journal on Optimization, Vol. 5, No. 1, pp. 192–218.

Fletcher R. and Leyffer S. (2002): *Nonlinear programming without a penalty function.* Mathematical Programming, Vol. 91, No. 2, pp. 239–269.

Fletcher R., Leyffer S. and Toint Ph. L. (2006): *A brief history of filter methods.* Technical Report ANL/MCS-P1372-0906, Mathematics and Computer Science Division, Argonne National Laboratory. Available at http://www.optimization-online.org/DB_FILE/2006/10/1489.pdf.

Franke R. (1994): *Anwendung von Interior-Point-Methoden zur Lösung zeitdiskreter Optimalsteuerungsprobleme.* M.S. thesis, Techniche Universität Ilmenau, Fakultät für Informatik und Automatsierung, Institut für Automatisierungs- und Systemtechnik Fachgebiet Dynamik und Simulation ökologischer Systeme, Ilmenau, Germany, (in German).

Franke R. (1998): *OMUSES – A tool for the optimization of multistage systems and HQP – A solver for sparse nonlinear optimization. Version 1.5*. Department of Automation and Systems Engineering, Technical University of Ilmenau, Germany. Available at ftp://ftp.systemtechnik.tu-ilmenau.de/pub/reports/omuses.ps.gz.

Franke R. and Arnold E. (1997): *Applying new numerical algorithms to the solution of discrete-time optimal control problems.* In: Computer–Intensive Methods in Control and Signal Processing: The Curse of Dimensionality, (Warwick K. and Kárný M., Eds.), Birkhäuser Verlag, Basel, Switzerland, pp. 105–118.

The solver Omuses/HQP for structured large-scale constrained optimization: Algorithm, implementation, and example application. Proceedings of the 6-th SIAM Conference on OPTIMIZATION, Atlanta.

Goldfarb D. and Idnani A. (1983): *A numerically stable dual method for solving strictly convex quadratic programs.* Mathematical Programming, Vol. 27, No. 1, pp. 1–33.

Gondzio J. (1994): *Multiple centrality corrections in a primal-dual method for linear programming.* Technical Report. 20, Department of Management Studies, University of Geneva, Geneva, Switzerland. Available at http://www.maths.ed.ac.uk/~gondzio/software/correctors.ps.

Griewank A., Juedes D., Mitev H., Utke J., Vogel O. and Walther A. (1999): *ADOL-C: A package for the automatic differentiation of algorithms written in C/C++, Version 1.8.2*, March 1999. Available at http://www.math.tu-dresden.de/~adol-c/.

Karmarkar N. (1984): *A new polynomial-time algorithm for linear programming.* Combinatorica, Vol. 4, No. 4, pp. 373–395.

Luus R. (2000): *Iterative Dynamic Programming*. CRC Press, Inc., Boca Raton, FL, USA.

Mehrotra S. (1992): *On the implementation of a primal–dual interior point method.* SIAM Journal on Optimization, Vol. 2, No. 4, pp. 575–601.

Misc J.-P. (2003): *Large scale nonconvex optimization.* SIAM's SIAG/OPT Newsletter Views-and-News, Vol. 14, No. 1, pp. 1–25. Available at http://fewcal.uvt.nl/sturm/siagopt/vn14_1.pdf.

Morales J. L., Nocedal J., Waltz R. A., Liu G. and Goux J.-P. (2001): *Assessing the potential of interior methods for nonlinear optimization.* Technical Report OTC 2001/4, Optimization Technology Center of Northwestern University.
Available at http://www.ece.northwestern.edu/~morales/PSfiles/assess.ps.

Nocedal J. and Wright S. J. (1999): *Numerical Optimization*. Berlin: Springer-Verlag.

De O. Pantoja J. F.A. (1988): *Differential dynamic programming and Newton's method.* International Journal of Control, Vol. 47, No. 5, pp. 1539–1553.

Powell M. J. D. (1978): *A fast algorithm for nonlinearly constrained optimization calculations.* In: Numerical Analysis, Dundee (G. A. Watson, Ed.), Dundee: Springer-Verlag, pp. 144–157.

Powell M. J. D. (1985): *On the quadratic programming algorithm of Goldfarb and Idnani.* Mathematical Programming Study, Vol. 25, pp. 46–61.

Salahi M., Peng J. and Terlaky T. (2005): *On Mehrotra-type predictor-corrector algorithms*. Technical report, Advanced Optimization Lab, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada. Available at http://www.optimization-online.org/DB_FILE/2005/03/1104.pdf.

Schittkowski K. (1980): *Nonlinear Programming Codes: Information, Tests, Performance*. Berlin: Springer-Verlag.

Schittkowski K. (1983): *On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search function.* Mathematishe Operations Forschung und Statistik, Ser. Optimization, Vol. 14, No. 2, pp. 197–216.

Schwartz A. and Polak E. (1997): *Family of projected descent methods for optimization problems with simple bounds.* Journal of Optimization Theory and Applications, Vol. 92, No. 1, pp. 1–31.

Tenny M. J., Wright S. J. and Rawlings J. B. (2002): *Nonlinear model predictive control via feasibility-perturbed sequential quadratic programming.* Technical Report TWMCC-2002-02, Texas-Wisconsin Modeling and Control Consortium. Available at http://jbrwww.che.wisc.edu/jbr-group/tech-reports/twmcc-2002-02.pdf.

Tits A. L., Wächter A., Bakhtiari S., Urban T. J. and Lawrence C.T. (2002): *A primal-dual interior-point method for nonlinear programming with strong global and local convergence properties.* Technical Report TR 2002-29, Institute for Systems Research, University of Maryland. Available at `http://www.ee.umd.edu/~andre/pdiprev.ps`.

Ulbrich M., Ulbrich S. and Vicente L. N. (2004): *A globally convergent primal-dual interior-point filter method for nonlinear programming.* Mathematical Programming, Vol. 100, No. 2, pp. 379–410.

Vanderbei R. J. and Shanno D. F. (1997): *An interior-point algorithm for non-convex nonlinear programming.* Technical Report SOR-97-21, Statistics and Operations Research, Princeton University. Available at `http://www.sor.princeton.edu/~rvdb/ps/nonlin.ps.gz`.

Wächter A. (2002): *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. Ph.D. dissertation, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA. Available at `http://www.research.ibm.com/people/a/andreasw/papers/thesis.pdf`.

Wächter A. and Biegler L. T. (2000): *Failure of global convergence for a class of interior point methods for nonlinear programming.* Mathematical Programming, Vol. 88, No. 3, pp. 565–574.

Wächter A. and Biegler L. T. (2005): *Line search filter methods for nonlinear programming: Motivation and global convergence.* SIAM Journal on Optimization, Vol. 16, No. 1, pp. 1–31.

Wächter A. and Biegler L. T. (2006): *On the implementation of a primal-dual interior-point filter line-search algorithm for large-scale nonlinear programming.* Mathematical Programming, Vol. 106, No. 1, pp. 25–57.

Waltz R. A. and Plantenga T. (2006): *KNITRO 5.0 User's Manual*. Available at `http://www.ziena.com/docs/knitroman.pdf`.

Wierzbicki A. (1984): *Models and Sensitivity of Control Systems*. Elsevier, Amsterdam.

Wright S. J. (1993): *Interior point methods for optimal control of discrete time systems.* Journal of Optimization Theory and Applications, Vol. 77, No. 1, pp. 161–187.

Wright S. J. (1997): *Primal-Dual Interior-Point Methods.* SIAM, Philadelphia, PA.

Yakowitz S. and Rutherford B. (1984): *Computational aspects of discrete-time optimal control.* Applied Mathematics and Computation, Vol. 15, No. 1, pp. 29–45.

Received: 21 March 2007
Revised: 8 July 2007